Universiteit Gent
Faculteit Ingenieurswetenschappen
Vakgroep Informatietechnologie

Dimensionerings- en werkverdelingsalgoritmen voor lambda grids

Dimensioning and Workload Scheduling Algorithms for Lambda Grids

Pieter Thysebaert

Proefschrift tot het bekomen van de graad van
Doctor in de Ingenieurswetenschappen:
Computerwetenschappen
Academiejaar 2006-2007

Universiteit Gent
Faculteit Ingenieurswetenschappen
Vakgroep Informatietechnologie

Promotoren: Prof. Dr. Ir. Bart Dhoedt
Prof. Dr. Ir. Filip De Turck

Universiteit Gent
Faculteit Ingenieurswetenschappen

Vakgroep Informatietechnologie
Sint-Pietersnieuwstraat 41, B-9000 Gent

Tel.: +32-9-331.49.19
Fax.: +32-9-331.48.99

Dit werk kwam tot stand in het kader van een mandaat van Aspirant bij het Fonds voor Wetenschappelijk Onderzoek - Vlaanderen (FWO-V).

Proefschrift tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen:
Computerwetenschappen
Academiejaar 2006-2007

Dankwoord

Ongelooflijk, maar opeens ben je aanbeland waar je jezelf nooit zag geraken: het afwerken van 4 jaar doctoraatsonderzoek en het schrijven van het bijhorende boek. Het begon allemaal in 2001, met het aanvragen van een onderzoeksbeurs bij het Fonds voor Wetenschappelijk Onderzoek onder impuls van mijn toenmalige thesisbegeleiders, Stefaan Vanhastel en Filip De Turck, die rond die tijd met het Grid concept in aanraking gekomen waren. Na goedkeuring van de beursaanvraag was ik vertrokken voor enkele jaren Gridonderzoek, een begrip waarvan de invulling gaandeweg evolueerde van een verzameling vage concepten en ideeën naar het werk en de resultaten beschreven in dit boek.

Hier wil ik dan ook even de tijd en ruimte nemen om de mensen waarmee ik tijdens deze toffe jaren in contact gekomen een welverdiende “15 characters of fame” te gunnen, aangezien elk van hen bijgedragen heeft tot het tot stand komen van dit werk. Vooreerst zou ik prof. Paul Lagasse wensen te bedanken aangezien dit werk onmogelijk geweest was indien ik geen gebruik had kunnen maken van de uitgebreide faciliteiten van de vakgroep Informatietechnologie. Uiteraard wil ik ook prof. Piet Demeester, prof. Bart Dhoedt, prof. Filip De Turck en dr. Stefaan Vanhastel bedanken om me de kans te geven dit doctoraatsonderzoek uit te voeren binnen de IBCN onderzoeksgroep alsook voor de talrijke begeleidende follow-ups en brainstormsessies.

Een speciale vermelding is er zeker voor (intussen dr.) Bruno Volckaert voor de vele goede samenwerkingsverbanden die we gesmeed hebben aangaande onderzoek, programmeren en publiceren - hierbij wens ik hem dan ook officieel aan te duiden als de belangrijkste medeschuldige voor de inhoud van dit boek. Andere Gridmensen die in het kader van dit onderzoek niet vrijuit gaan zijn Marc De Leenheer, Maria Chtepen en Jürgen Baert.

Mijn geworstel met wereldlijke problemen (zoals reisaanvragen, vliegtuigtickets, SAP, weekstaten en koffiekaarten) werd draaglijker gemaakt door Martine Buysse, Ilse Van Royen, Davinia Stevens, Marleen Van Duyse, Karien Hemelsoen, Ilse Meersman en Bernadette Becue.

Ook de mensen die onze IT-infrastructuur draaiende houden wil ik speciaal bedanken. Op vragen omtrent TwinTech, MPL, RAID, 3Ware, BEGrid en stroompannes hadden dr. Brecht Vermeulen, Bert De Vuyst, Wouter Adem, Pascal Vandeputte en Stijn De Smet steeds gezwind een antwoord klaar.

Verder zijn er ook nog een resem mensen die mij bureaugewijs getolereerd hebben doorheen de jaren: Frederik Scholaert, Bart Puype, Thijs Lambrecht (bedankt voor de humor), dr. Steven Van den Berghe (bedankt voor de GUIs), Koen De Proft, Kristof Taveirne, Stijn Eeckhaut, Johannes Deleu.

Hielpen mij streng maar rechtvaardig optreden tijdens de practicumsessies: Andy Van Maele, Tom Van Leeuwen en Jeroen Hoebeke.

Naast het onderzoeks- en practicumwerk heeft IBCN mij ook de mogelijkheid geboden om mijn sportieve capaciteiten aan te scherpen. Mijn sterk verbeterde oog-hand-coördinatie komt dan ook op het conto van Bartman (dr. Bart Duysburgh), CraHan (Thomas Bouve), mezelf (Tom Verdickt), JC (dr. Jan Coppens), Walter Capiau (dr. Erik Van Breusegem) en Pee (dr. Peter Backx). Denksporttraining kreeg ik van Tim Wauters, dr. Jan Cheyns, Filip De Greve, Tim Stevens, Bruno Vandenbossche en Nico Goeminne. Mijn steile opgang in de badmintonratings wijt ik terloops aan dr. Chris Develder.

Om de weinige vrije tijd die me restte buiten de werk- en sporturen toch nog enigszins nuttig op te vullen (m.b.v. de nodige frisdrank en versnaperingen) kon ik bovendien steeds rekenen op Sofie Van Hoecke, Stefan Bouckaert, Koert Vlaeminck, Benoît Latré, Sarah Vullers, Frederic Van Quickenborne en ons eigen wandelend evenementenbureau Bart Lannoo.

Tenslotte zou ik graag nog mijn ouders en broer willen bedanken voor de vele hulp en onvoorwaardelijke steun die ik al die tijd mocht ontvangen en bij wie ik steeds terecht kon.

Hopelijk ben ik niet teveel mensen vergeten - indien wel wil ik ook u, de niet bij naam genoemde lezer, bedanken voor de interesse in dit werk.

Gent, augustus 2006
Pieter Thysebaert

Table of Contents

List of Figures

List of Tables

List of Acronyms

B

BDP	Bandwidth-Delay Product

C

CASTOR	CERN Advanced Storage Manager
CBR	Constant Bit Rate
CERN	Conseil Européen pour la Recherche Nucléaire
CR	Computational Resource
CWDM	Coarse Wavelength Division Multiplexing

D

DLT	Divisible Load Theory
DR	Data Replica Resource
DWDM	Dense Wavelength Division Multiplexing

E

EGEE	Enabling Grids for E-Science in Europe

F

FCFS	First-Come, First-Served

G

GGF	Global Grid Forum
GLIF	Global Lambda Integrated Facility
GSI	Grid Security Infrastructure
GTP	Group Transport Protocol

H

HEP	High Energy Physics

I

IAT	Inter-Arrival Time
ILP	Integer Linear Program
IP	Internet Protocol

L

LCG	Large Hadron Collider Computing Grid
LHC	Large Hadron Collider
LP	Linear Program

M

MMRCPSP	Multi-Modal Resource Constrained Project Scheduling Problem
MTBF	Mean Time Between Failures
MTTF	Mean Time To Fail
MTTR	Mean Time To Repair

N

ns-2	Network Simulator 2

O

OBS	Optical Burst Switching
OCS	Optical Circuit Switching
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
OPS	Optical Packet Switching
OTN	Optical Transport Network
OXC	Optical Cross Connect

Q

QoS	Quality of Service

R

RBUDP	Reliable Blast User Datagram Protocol
RCPSP	Resource Constrained Project Scheduling Problem
RFIO	Remote File Input/Output

S

SABUL	Simple Available Bandwidth Utilization Library
SR	Storage Resource

T

TCP	Transport Control Protocol
TOS	Type of Service

U

UDP	User Datagram Protocol

V

VO	Virtual Organization
VPG	Virtual Private Grid
VPN	Virtual Private Network

W

WDM	Wavelength Division Multiplexing
WSRF	Web Services Resource Framework

X

XSLT	eXtensible Stylesheet Language – Transformations
XML	eXtensible Markup Language

Samenvatting

De alomtegenwoordigheid van het Internet en van (vaak onderbenutte) rekenkracht heeft aan het eind van het vorige decennium aanleiding gegeven tot het ontstaan van de idee van een schier onuitputtelijke bron van rekenkracht, op alle tijdstippen beschikbaar voor haar gebruikers. Naar analogie met de beschikbaarheid en toegankelijkheid van het lichtnet (Eng.: *Power Grid*), werd voor dit concept van onuitputtelijke en op aanvraag verkrijgbare rekenkracht algauw de term *Grid* in gebruik genomen.

Een dergelijke Grid wordt gevormd door de verenigde krachten van meerdere heterogene bronnen die zich mogelijk bevinden op verscheidene geografische locaties, waarbij gespecialiseerde *middleware* gebruikt wordt om de toegang tot het geheel en de coördinatie tussen de verschillende bronnen onderling te verzorgen. Belangrijke brontypes zijn de computationele elementen (die over processorkracht beschikken), data-opslagelementen en de netwerkverbindingen die de overige bronnen in staat stellen te communiceren en zodoende van het geheel effectief een gedistribueerd computersysteem maken.

Door de verspreiding van de bronnen, en dus door het noodzakelijkerwijs gebruiken van lange-afstandsnetwerkverbindingen vormt zo'n Grid een eerder zwak gekoppeld gedistribueerd computersysteem. De totale verwerkings- en opslagcapaciteit ervan kunnen echter moeiteloos die van een sterk gekoppeld multiprocessorsyteem overtreffen, waarbij dit laatste type systeem het nadeel vertoont veel sneller in kostprijs toe te nemen bij stijgende totale verwerkingscapaciteit.

Het succes van een Grid als gedistribueerd computersysteem wordt mede bepaald door de applicaties die ervan gebruik kunnen maken. In de wetenschappelijke wereld zijn dergelijke applicaties echter legio; het applicatie-type dat bij uitstek geschikt is voor uitvoering in een Gridomgeving omvat de zogenaamde *parameter sweep* applicaties, aangezien dergelijke applicaties kunnen opgesplitst worden in een groot aantal quasi-onafhankelijke deeltaken. Bekende Grid-gebruiksscenario's zijn o.m. het distribueren en analyseren van de enorme hoeveelheid gegevens (grootteorde PB/jaar) die jaarlijks gegenereerd worden aan het CERN, het collaboratief onderzoek verricht binnen het EScience Grid project en de wijdverspreide *cycle stealing* applicaties zoals SETI@Home, waarbij de Grid in feite bestaat uit de verzameling desktop PC's (verbonden via het Internet) van eenieder die de betreffende applicatie geïnstalleerd heeft. Aan de andere kant van het Grid-spectrum waarbij op grote hoeveelheden data complexe analyses uitgevoerd dienen te worden ontstaat een trend om de verschillende bronnen binnen de Grid te connecteren via optische (circuitgeschakelde) transportnetwerken die in staat zijn om zeer hoge

bandbreedtes te garanderen.

Waar reeds veel onderzoeksresultaten beschikbaar zijn voor het dimensioneren van netwerken (uitgaande van een trafiekmatrix) en de bronallocatie en werklastverdeling binnen een clusteromgeving, kunnen deze resultaten niet zomaar overgenomen worden voor de evaluatie van de werking van een Gridomgeving. Typerend voor de werking van Grid is de noodzaak om meerdere bronnen tegelijk te reserveren voor de uitvoering van een applicatie (typisch computationele en netwerkbronnen), terwijl men in een lokale cluster vaak het interconnectienetwerk en/of de hoeveelheid trafiek erover niet in beschouwing dient te nemen. De trafiek op het Grid-interconnectienetwerk hangt bovendien af van de exacte werkverdelingsstrategie die gebruikt wordt; deze strategie heeft dus onmiskenbaar invloed op de dimensioneringsproblematiek van de Grid en het gebruikte interconnectienetwerk.

Het werk beschreven in dit boek kan opgesplitst worden in de hierna beschreven onderdelen. Na een korte inleiding wordt vooreerst een overzicht gegeven van de belangrijkste concepten die relevant zijn voor het werk in dit boek. Naast de technologische aspecten van Grids gaat het hierbij vooral om technieken en hulpmiddelen die toelaten om werklast te verdelen over de verschillende bronnen in dergelijke Grids en de betreffende bronnen correct te dimensioneren. Voorts wordt ook aandacht besteed aan concepten en hulpmiddelen die gebruikt worden om statusgegevens en prestatiemetrieken uit operationele Grids te extraheren, alsmede hulpmiddelen om de werking van Grids correct te modelleren en simuleren.

In het kader van dit onderzoekswerk werd zo'n Grid-simulatieomgeving ontwikkeld. De modellen voor de verschillende Gridentiteiten die in deze omgeving geïmplementeerd werden (deze omvatten zowel modellen voor de Gridapplicaties als voor de verschillende types systeembronnen) alsook de opbouw en werking van deze omgeving vormen een hoofdstuk op zich.

Vervolgens worden resultaten in twee verschillende toepassingsdomeinen, verkregen met de voornoemde Grid-simulatieomgeving, voorgesteld. In een eerste toepassing wordt aangetoond hoe het behandelen van netwerkbronnen als volwaardige systeembronnen bij het verdelen van werklast in een Grid een verbetering meebrengt m.b.t. de gemiddelde verwerkingstijd. Een tweede toepassing omhelst het partitioneren van de systeembronnen binnen een Grid over verschillende applicatietypes om zodoende een verzameling Virtual Private Grids (VPGs) te creëren. We vergelijken een tweetal verschillende partitioneringsmethodes, opnieuw aan de hand van gemiddelde verwerkingstijden voor de Gridapplicaties.

Doordat Gridapplicaties typisch grote hoeveelheden data dienen te verwerken is er bijzondere interesse binnen de Gridgemeenschap om verschillende Gridsites te verbinden met een optisch transportnetwerk waarbij data over verschillende golflengtes verstuurd wordt. Dergelijke netwerken hebben immers het voordeel grote hoeveelheden data aan hoge snelheid te kunnen transporteren zonder overgevoelig te zijn voor storingsinvloeden. Een belangrijk vraagstuk binnen dergelijke zogenaamde lambda Grids betreft de installatie van de nodige netwerkcapaciteit in functie van de Gridwerking.

In een volgend hoofdstuk wordt dan ook gekomen tot een modellering van

het dimensioneringsprobleem van Grids die gebruik maken van circuitgeschakelde optische transportnetwerken. Dit model neemt de vorm aan van een lineair programma waarvan door achtereenvolgende ingrepen de complexiteit sterk verminderd werd vergeleken met de technieken die in het overzichtshoofdstuk besproken werden. Deze dimensioneringstechniek spitst zich toe op de rekenbronnen en netwerkelementen in de Grid, en werd geëvalueerd voor verschillende operationele scenario's en technologieparameters. Waar eerst wordt uitgegaan van volledig betrouwbare bronnen en netwerkelementen, worden ook scenario's bestudeerd waarin rekening gehouden wordt met mogelijk falende bronnen en netwerkelementen. Waar mogelijk worden - door middel van analytische afleiding - de bekomen resultaten vergeleken met resultaten bekomen voor reguliere netwerktopologieën.

In het laatste hoofdstuk wordt het verdelen van werklast in een operationele Grid bekeken. In tegenstelling tot het voorgaande hoofdstuk ligt de nadruk hier dus op zgn. *on-line* technieken, terwijl dimensioneringsmethoden noodzakelijkerwijs dienen te gebeuren vóór de inbedrijfname van de Grid. De bestudeerde technieken bouwen echter wel voort op de resultaten van de dimensionering die in het vorige hoofdstuk gebeurde door de daar berekende *off-line* werkverdeling te gebruiken als richtpunt. Verschillende van deze technieken werden bestudeerd waarbij onderscheid gemaakt werd in functie van de gebruikte kostfuncties en prestatiemetrieken, en een vergelijking met standaard werkverdelingsmethoden (niet expliciet steunend op de oplossing van het dimensioneringsprobleem) werd gemaakt.

Dit boek wordt afgesloten met een overzicht van de belangrijkste bijdragen geleverd in dit werk en de bijhorende conclusies.

Summary

One decade ago, the Internet's omnipresence and the large amount of idle processing power it harnesses spawned the vision of a vast and nigh inexhaustible well of computing power featuring round-the-clock availability. As one cannot help but note the similarity of this concept to the always available and easy to use Power Grid, this envisioned computing platform has become known as the Grid.

Such a Grid consists of the conglomeration of multiple heterogeneous resources, possibly geographically dispersed. These different resources are accessed and managed through specialized Grid *middleware*. The most important resource types in a Grid are the computational resources (providing processing power), data storage resources and the network elements responsible for the interconnection of all resources, effectively turning the Grid into a distributed computing platform.

As Grid resources may be scattered across the globe, the resulting Grid behaves more or less like a loosely coupled distributed computer system in which long-haul network connections are plenty. The aggregate processing and storage capacities of this system can, however, easily surpass those of a tightly coupled multiprocessor system, while this latter type of system typically exhibits a steeper price/performance ratio with increasing processing capacity.

Ultimately, the Grid's success as a universally accepted distributed computing platform is determined to a great extent by the applications it supports. Today, Grid applications can be found in abundance primarily in the scientific communities. Many of these applications perform parameter sweeps over some domain; this kind of application can easily be separated into multiple quasi-independent parts which can then be distributed over multiple computing resources.

The best-known use cases of a Grid include the distribution and analysis of data generated at CERN's LHC (magnitude: several PB/year), the collaborative research environment boasted by the EScience Grid project and the numerous *cycle-stealing* applications of which SETI@Home is a prime example (in this case, the Grid in question consists of a collection of idle desktop PCs connected through the Internet).

In order to support Grid applications processing large amounts of data (e.g. the CERN case), optical networking technologies have received a great deal of interest from the Grid community. These optical transport networks can provide very high data rates without significant drawbacks in terms of reliability.

Most relevant research regarding the dimensioning of such networks (from given demand matrices), the resource allocation and the workload scheduling policies cluster or multiprocessor setups cannot simply be transferred into the Grid

realm and applied as such. Indeed, a defining property of a Grid is the need for simultaneous co-allocation of multiple resources (e.g. both computational and network resources) to each application, while in a local cluster setting one can regularly disregard the local interconnecting network's influence and assume it to be sufficiently performant, that is, suffering only from a minimal delay and offering quasi-infinite bandwidth. In addition, in a Grid environment the network traffic generated is closely tied to the workload distribution policy used. Thus, the Grid dimensioning problem also differs significantly from the problem of dimensioning optical transport networks from static demand matrices, as the workload distribution policy cannot be disregarded when studying the former problem.

The research described in this book consists of the following parts. Starting with an introductory section, we continue with an overview of the key concepts relevant to our research. This includes both the technological aspects of Grids as well as techniques and tools used to distribute workload over the various resources in a Grid and to dimension these resources accordingly. In addition, we describe the major tools and concepts used in Grid monitoring (during which performance metrics and resource status data are gathered), Grid modeling and Grid simulation.

During the course of this research such a Grid simulation environment has been developed. The Grid application and resource models implemented in this simulation tool, as well as its structure and mode of operation have been detailed in a dedicated chapter. We present results from two different applications of our Grid simulation environment. In a first application, we show how network aware workload scheduling in Grids can significantly improve application response times when compared to non-network aware scheduling. In a second application, we focus on partitioning the Grid's resources into multiple Virtual Private Grids (VPGs). We compare two different Grid partitioning approaches, again using the average application response time as metric.

As Grid applications typically need to process large amounts of data, a lot of interest has been sparked within the Grid research community to connect different Grid sites using an optical transport network, in which data is transported over multiple wavelength paths. These networks are able to transport large amounts of data at extremely high bitrates while not being excessively prone to interference. In these so-called lambda Grids, an important problem is the installation of sufficient network capacity, given the Grid's mode of operation.

The book's next chapter features a model of this optical circuit switched lambda Grid dimensioning problem. This model comes as a linear program, which has been greatly reduced in complexity by applying multiple improvements over the techniques mentioned in the overview chapter. We focus primarily on computational resources and network elements and evaluate our approach for different Grid scenarios and technology-related parameter variations. We start with scenarios featuring completely dependable resources and extend our approach to scenarios featuring possible resource failures. By means of analytical derivation, we compare our results to results obtained for regular network topologies.

Lastly, we deal with the issue of workload distribution in an operational Grid. While the previous chapter focusses on off-line dimensioning techniques, the core

contribution here concerns on-line scheduling techniques. The algorithms we present do make use of the results obtained during the off-line dimensioning, however. In particular, they use the then-calculated optimal off-line workload distribution as a target. Several of these algorithms (differing in their use of cost functions and performance metrics) were evaluated against a suite of standard workload scheduling algorithms (which do not make use of this off line calculated target).

Finally, we summarize our main research contributions and present the major conclusions to be drawn.

1 Introduction

1.1 The Grid Concept

Traditionally, supercomputers and clusters have been the platforms of choice for solving computationally complex problems, implemented as applications featuring multiple parallel (communicating or independent) tasks. However, demands for computational processing power are only increasing. Unfortunately, supercomputers feature a superlinear increase of price with growing processing capacity, and local clusters are confined to a single room or building.

If it is possible, however, to simultaneously utilize multiple idle resources scattered over different locations, the resulting aggregate computing power can easily surpass that of any single supercomputer or cluster. Such a construction is coined "Grid" [1, 2], in an analogy with the electrical power grid, as it is deemed to provide us with the notion of ubiquitous computing power.

The successful operation of such a Grid faces many challenges, as different resources are administered and managed using different policies and may join or leave the Grid in an unpredictable way. Furthermore, interconnecting such resources through a possibly insecure network means that the necessary software infrastructure must be present to perform authentication and authorization actions for users participating in a collaborative environment sharing resources among each other (so-called *Virtual Organizations* (VOs) [1]). The Globus Toolkit [3] is a middleware toolkit aimed at solving these and other challenges identified by the Global Grid Forum [4], the body incorporating individuals from research and

industrial sectors concerned with Grid standardization efforts.

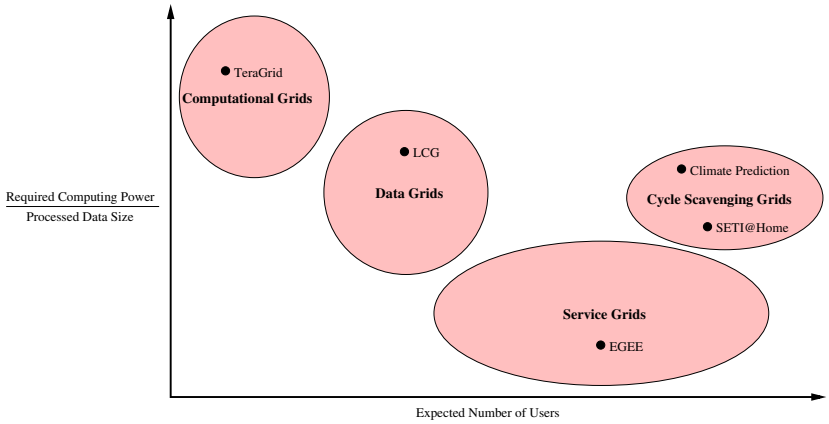


Figure 1.1: Grid Taxonomy: Classification by Resource Requirements and number of Users

Different types of Grids have been identified in [5]. Computational Grids support applications requiring lots of processing power when compared to the available bandwidth to storage facilities they need. The US-based TeraGrid [6] is a famous example in this class. It consists of 8 sites connected through a high-speed network and, as the project name suggests, each site offers several teraFLOPS of processing capacity and tens to hundreds of terabytes of storage capacity. Another notable Grid class contains the *Cycle-stealing* applications; they make use of idle desktop PCs and present themselves as screensavers. Notable examples of such applications include *Seti@Home* [7], *Climate Prediction* [8] (both employing the *BOINC* [9] cycle-stealing software framework) and the *Screensaver-Lifesaver* [10] anti-cancer drug research effort. In Data Grids, not only a large amount of computational power needed is needed but also large data sets need to be moved around in a timely fashion as well, increasing the importance of network resources (or, more accurately, the co-allocation of network and computing resources). The prime example in this area is the LCG project [11], in which petabytes of data generated by CERN's Large Hadron Collider are distributed for processing following a multi-tier model.

Service Grids (such as the Enabling Grids for E-Science in Europe (EGEE) project [12], to which the Belgian BEGrid [13] is hooked up) generally denote any Grid offering services beyond the capabilities of a single machine. These services could include collaborative working environments or multimedia processing Grids and as such include facilities to support real-time application needs.

A possible classification of different Grid types has been visualized in figure 1.1.

Initially, Grid research focused mainly on computationally complex problems. The emergence of Data Grids (such as the LCG [11] project), however, has generated an increased interest in multi-resource QoS research. This interest is only emphasized further by the distributed nature of Grids, rendering the interconnecting network between Grid sites a non-negligible resource. In addition, the large amounts of data processed and transferred in such a Grid have led to the investigation of the suitability of optical transport networks in this Grid context. It is this type of Grid that is studied in our work in greater detail.

1.2 Problems and Challenges

As the deployment and use of Grids as distributed computing environments diverges from the localized high-throughput computing cluster, several new challenges arise. First of all, due to the large number of resources, users and jobs, a scalable resource management and allocation infrastructure is required. Secondly, due to the large number of resources involved and their respective use policies, (temporary) resource failure or unavailability is bound to happen, giving rise to the Grid's typical dynamic nature. Therefore, resilience to these resource failure scenarios is an important design criterion for Grids. In addition, an important property of Grids is their distributed (i.e. geographically dispersed) nature. This implies that it is important that the network interconnecting the various Grid sites is treated as a first-class resource i.e. of equal importance when compared to e.g. computational and storage resources. As resource usage ultimately depends on the Grid's workload and the deployed resource selection and allocation policy, it can be concluded that the issues raised here not only impact a Grid's scheduling and resource management policies, but also have their influence on each resource's capacity decided upon when dimensioning the Grid prior to its deployment. This dimensioning problem is complicated further by the increasing use of optical transport networks in Grids which enforce additional network constraints. Finally, these concerns - applicable to Grid deployment and implementation - need to be coped with when modeling and simulating an operational Grid in advance as well. As such, it is safe to state that the modeling and solving of the Grid dimensioning and resource allocation problems differ enough from the localized, computing oriented cluster case to warrant dedicated research efforts.

1.3 Main Research Contributions

In the first phase of our work, a Grid simulation environment was implemented. This simulation environment contains detailed Grid application and resource models (including computational, storage and network resource models) as well as

the necessary software component models for the Grid middleware, in particular scheduling, resource management and resource information components. Of particular interest, and directly related to the work described in this book, is the fact that the scheduling component is highly extensible in order to allow newly developed scheduling policies to be "plugged in" easily.

The models mentioned have been re-used to generate a scalable linear programming formulation of the steady state Grid dimensioning and scheduling problems. Special attention has been paid to so-called *lambda Grids*, where the various sites are interconnected through an optical transport network. These optical transport networks lead to additional complexity in the linear programming formulation because they require bandwidth granularity and wavelength continuity constraints.

Because of the dynamic nature of Grids, this formulation has been extended to cope with single-resource failures in lambda Grids - specifically computational resource, network link and optical cross-connect failures have been taken into account.

This linear programming approach has also been used to evaluate partitioning strategies used to allocate adequate resources to different VOs, as Grid sites may participate in multiple VOs. The resulting sub-Grids are denoted Virtual Private Grids. We have compared our linear programming based VO partitioning strategy to an approach using a genetic algorithm by means of simulation in our Grid simulation environment.

Based on the off-line solution to the steady state Grid dimensioning and scheduling problems, we have devised several on-line workload scheduling algorithms. These algorithms have been carefully constructed to deal with multiple different resource types in a sensible way, and have been evaluated in our Grid simulation environment.

A Grid monitoring architecture has been developed simultaneously. Such a monitoring architecture is able to gather relevant resource and application data from an operational Grid, which can then be fed back into the deployed scheduling algorithms, both in the operational Grid and in its simulated counterpart. Our implementation of this monitoring framework is geared towards scalability and performance and has been compared extensively to established Grid monitoring and information frameworks.

The Grid simulation environment and Grid monitoring architecture mentioned here have been co-developed with Bruno Volckaert. He has used these software platforms primarily to study several Grid software architectures. For instance, he has profiled and evaluated a scalable Grid partitioning architecture (by implementing it in NSGrid), its algorithms and its deployment in service and media Grids. The results of this evaluation have been described in his PhD thesis titled "Architectures and Algorithms for Network and Service Aware Grid Resource Management". The NSGrid implementations of the algorithms for Grid dimensioning

and workload distribution described in this thesis have been carried out by Pieter Thysebaert.

1.4 Outline

This thesis is structured as follows: chapter 2 presents the relevant Grid research context and sets the scene for the following chapters. In particular, this chapter highlights current Grid deployments, available middleware and the growing importance of optical interconnection networks in Grids. Modeling techniques for dimensioning and scheduling problems in traditional distributed environments are given; special attention is paid to the requirements imposed by a Grid environment on these modeling techniques.

Chapter 2 also details the required properties of successful Grid Monitoring systems and discusses some important examples as well as areas in which these systems can be improved. Finally, this chapter discusses and compares some important Grid simulation tools and exposes their merits and limitations.

In chapter 3, the Grid simulation environment developed during the course of our research and the models it implements are detailed, and we draw attention to some important features that distinguish our simulation environments from existing alternatives listed in chapter 2.

Chapter 4 demonstrates the application of NSGrid to two separate use cases.

Chapter 5 specifically deals with the off-line dimensioning of lambda Grids. To solve this dimensioning problem, we extend the modeling techniques featured in chapter 2 and adapt them to suitable Grid dimensioning techniques. In particular, resource allocation interdependencies and scalability are key issues dealt with in our problem modeling. We present dimensioning results using our model for two-tier Grids.

Our research on on-line Grid scheduling techniques makes up chapter 6. We describe several scheduling algorithms designed to distribute workload in a steady state Grid, using concepts introduced in chapter 5. We evaluate the effectiveness of these algorithms on Grids dimensioned using the technique developed in chapter 5. The software platform used to evaluate the algorithms is the Grid simulation environment detailed in chapter 3.

We end this thesis with some concluding remarks in chapter 7.

1.5 Publications

1.5.1 Publications in international journals

- **P. Thysebaert**, B. Volckaert, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *Scalable Dimensioning of Resilient Lambda Grids*, submitted to

Future Generation Computer Systems - The International Journal of Grid Computing: Theory, Methods and Applications.

- F. De Turck, J. Decruyenaere, **P. Thysebaert**, S. Van Hoecke, B. Volckaert, C. Danneels, K. Colpaert, G. De Moor, *Design of a Flexible Platform for Execution of Medical Decision Support Agents in the Intensive Care Unit*, to appear in Elsevier Journal of Computers in Biology and Medicine, 2006

- **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, *Evaluation of Grid scheduling strategies through NSGrid: a network aware Grid simulator*, published in Neural, Parallel & Scientific Computations, Special Issue on Grid Computing, Dynamic Publishers Atlanta, Editors H.R. Arabnia, G.A. Gravvanis, M.P. Bekakos, 12(3):353-378, 2004.

- B. Volckaert, **P. Thysebaert**, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *Grid computing: the next network challenge!*, published in The Journal of The Communications Network, Proceedings of FITCE 2004, 43rd European Telecommunications Congress, 3:159-165, 2004.

- B. Volckaert, **P. Thysebaert**, F. De Turck, B. Dhoedt, P. Demeester, *Application-specific hints in reconfigurable Grid scheduling algortihms*, published in Lecture Notes in Computer Science, Proceedings of ICCS 2004, Springer-Verlag Berlin Heidelberg, Krakow, LNCS 3038:149-157, 2004.

- **P. Thysebaert**, B. Volckaert, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *Resource partitioning algorithms in a programmable service Grid architecture*, published in Lecture Notes in Computer Science, Proceedings of the 5th Intern. Conf. on Computational Science ICCS 2005, Atlanta, LNCS 3516:250-258, 2005.

- **P. Thysebaert**, M. De Leenheer, B. Volckaert, B. Dhoedt, P. Demeester, *Scalable Dimensioning of Optical Transport Networks for Grid Excess Load Handling*, accepted for publication in Photonic Network Communications (PNC), 2006.

- M. De Leenheer, **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, D. Simeonidou, R. Nejabati, G. Zervas, D. Klonidis, M. J. OMahony, *A View on Enabling Consumer Oriented Grids through Optical Burst Switching*, published in IEEE Communications Magazine, 44(3):124-131, 2006.

- B. Volckaert, **P. Thysebaert**, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *Flexible Grid service management through resource partitioning*, accepted for publication in the Journal of Supercomputing, 2006.

- **P. Thysebaert**, B. Volckaert, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *Dimensioning and On-Line Scheduling in Lambda Grids using Divisible Load Concepts*, accepted for publication in the Journal of Supercomputing, 2006.

1.5.2 Chapters in international publications

- B. Volckaert, **P. Thysebaert**, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *Network and Service Aware Grid Resource Assignment*, to be published as a chapter in Grid Technologies: Emerging from Distributed Architectures to Virtual Organizations, Editors: M.P. Bekakos, G.A. Gravvanis and H.R. Arabnia, WIT Press.

1.5.3 Publications in international conferences

- S. Vanhastel, **P. Thysebaert**, F. De Turck, B. Volckaert, P. Demeester, B. Dhoedt, *Service brokering in an enhanced grid environment*, published in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, 2:712-718, 2002.

- F. De Turck, S. Vanhastel, **P. Thysebaert**, B. Volckaert, P. Demeester, B. Dhoedt, *Design of a middleware-based cluster management platform with task management and migration*, published in 2002 IEEE International Conference on Cluster Computing and the Grid, Chicago, pages 484-487, 2002.

- B. Volckaert, **P. Thysebaert**, F. De Turck, P. Demeester, B. Dhoedt, *Evaluation of grid scheduling strategies through a network-aware grid simulator*, published in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'03, Las Vegas, 1:31-35, 2003.

- **P. Thysebaert**, B. Volckaert, M. De Leenheer, E. Van Breusegem, F. De Turck, B. Dhoedt, D. Simeonidou, M.J. O'Mahony, R. Nejabati, A. Tzanakai, I. Tomk, *Towards consumer-oriented photonic grids*, published and presented at Workshop on Optical Networking for Grid Services at ECOC2004 - on CD-ROM, Stockholm, 2004.

- B. Volckaert, **P. Thysebaert**, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *On the use of NSGrid for accurate grid schedule evaluation*, published in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'04, Las Vegas, 1:200-206, 2004.

- B. Volckaert, **P. Thysebaert**, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *Network aware scheduling in grids*, published in Proceedings of NOC2004, 9th European Conference on Networks & Optical Communications, Eindhoven, pages 311-318, 2004.

- **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, *Network aspects of grid scheduling algorithms*, published in Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems, San Francisco, pages 91-97, 2004.

- M. De Leenheer, **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, *Evaluation of a job admission algorithm for bandwidth constrained grids*, published in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'04, Las Vegas, 2:591-594, 2004.

- M. De Leenheer, E. Van Breusegem, **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, D. Simeonidou, M.J. O'Mahony, R. Nejabati, A. Tzanakaki, I. Tomkos *An OBS-based grid architecture*, published in 2004 IEEE Globecom Telecommunications Conference Workshops, Dallas, pages 390-394, 2004.

- S. De Smet, **P. Thysebaert**, B. Volckaert, M. De Leenheer, D. De Winter, F. De Turck, B. Dhoedt, P. Demeester, *A performance oriented grid monitoring architecture*, published in Proceedings of the 2nd IEEE Workshop on End-to-End Monitoring Techniques and Services (E2EMON), Monitoring Emerging Network Services, San Diego, pages 23-28, 2004.

- **P. Thysebaert**, F. De Turck, B. Dhoedt, P. Demeester, *Using Divisible Load theory to Dimension Optical Transport Networks for Grid Excess Load Handling*, published in Proceedings (on CD-ROM) of the 2005 Optical Fiber Communications Conference and Exposition, Anaheim, 2005.

- B. Volckaert, **P. Thysebaert**, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *A Distributed Resource and Network Partitioning Architecture for Service Grids*, published in Proceedings of (on CD-ROM) the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid05), Cardiff, 2005.

- F. Farahmand, M. De Leenheer, **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, J.P. Jue, *A multi-layered approach to optical burst-switched based grids*, published in Proceedings (on CD-ROM) of Workshop on Optical Burst/packet Switching (WOBS2005), 2nd International Conference on Broadnet Net, Boston, pages 127-134, 2005.

- **P. Thysebaert**, M. De Leenheer, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, *Using Divisible Load Theory to Dimension Optical Transport Networks for Grid Excess Load Handling*, published in Proceedings (on CD-ROM) of the International Conference on Networking and Services (October 2005)

- M. De Leenheer, F. Farahmand, **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, J. Jue, *Anycast routing in optical burst switched grid networks*, published in Proceedings of ECOC2005, 31st European Conference on Optical Communications, Glasgow, 3:699-702, 2005.

- B. Volckaert, **P. Thysebaert**, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester, *A scalable and performant grid monitoring and information framework*, published in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '05, Las Vegas, 1:224-230, 2005.

- J. Baert, M. De Leenheer, B. Volckaert, T. Wauters, **P. Thysebaert**, F. De Turck, B. Dhoedt, P. Demeester, *Hybrid optical switching for data-intensive media grid applications*, to be published in Proceedings of the workshop on Design of Next Generation Optical Networks: from the Physical up to the Network Level Perspective, Gent, 2006.

1.5.4 Publications in national conferences

- **P. Thysebaert**, B. Volckaert, F. De Turck, S. Vanhastel, P. Demeester, *Management of Network Resources in a Grid-Computing Environment*, published in 2nd FTW PHD Symposium, Interactive poster session, Ghent, paper nr. 72, 2001.

- B. Volckaert, **P. Thysebaert**, F. De Turck, B. Dhoedt, P. Demeester, *A generic grid simulator for evaluating network-aware grid scheduling algorithms*, published in 3rd FTW PHD Symposium, Interactive poster session, Ghent, paper nr. 21, 2002.

- M. De Leenheer, E. Van Breusegem, J. Cheyns, **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, *Optical burst switching for consumer grids*, published in 5th FTW PHD Symposium, Interactive poster session, Ghent, paper nr. 102, 2004.

- **P. Thysebaert**, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, *Grid scheduling and dimensioning using divisible load theory*, published in 5th FTW PHD Symposium, Interactive poster session, Ghent, paper nr. 123, 2004.

References

[1] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[2] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure 2nd Edition*. Morgan Kaufmann, 2003.

[3] I. Foster. *Globus Toolkit Version 4: Software for Service-Oriented Systems*. Lecture Notes in Computer Science, 3779:2–13, 2005.

[4] *Global Grid Forum*. `http://www.gridforum.org/`.

[5] K. Krauter, R. Buyya, and M. Maheswaran. *A Taxonomy and Survey of Grid Resource Management Systems*. International Journal of Software: Practice and Experience (SPE), 32:135–164, 2002.

[6] *The TeraGrid project*. `http://www.teragrid.org/`.

[7] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. *SETI@home: An Experiment in Public-Resource Computing*. Communications of the ACM, 45:56–61, 2002.

[8] M. Allen. *Do it yourself climate prediction*. Nature, 401:642, 1999.

[9] *Berkeley Open Infrastructure for Network Computing*. `http://boinc.berkeley.edu/`.

[10] W. Graham Richards. *Virtual Screening using Grid Computing: the Screensaver Project*. Nature Reviews Drug Discovery, 1:551–558, 2002.

[11] *LHC Computing Grid project*. `http://lcg.web.cern.ch/LCG`.

[12] *Enabling Grids for E-Science in Europe*. `http://egee-intranet.web.cern.ch`.

[13] *Belnet Grid Initiative*. `http://www.begrid.be/`.

2 Research Context

2.1 Introduction

The idea of a Grid is the idea of a dependable, inexpensive and easily accessible (distributed) computing environment, resembling the way electricity is distributed over modern power grids. Key to the construction of such an environment is the coupling of existing building blocks, such as computational clusters, storage solutions and networks. Upon successful deployment, a Grid will feature good utilization of these resources (a great deal of which are idle during a lot of time), and secure and transparent access to these resources to create the possibility of a truly collaborative working environment.

In this chapter, we highlight the essential components and concepts that make up a Grid, give an overview of relevant research activities concerned with each of these concepts and indicate how these concepts form the starting point for our research contributions. We start with an overview of vital Grid-enabling software components in sections 2.2 and 2.3. Next, we focus on the state of optical networking technologies and their use in lambda Grids in section 2.4. The modeling of the optical network dimensioning problem from static demand matrices is covered in section 2.5; in a lambda Grid setting, this problem must be extended as demands depend on the workload distribution used. This extended problem is the basis for our work presented in chapter 5. As the lambda Grid dimensioning problem depends on the exact workload schedule, the off-line modeling of a workload scheduling problem is discussed in section 2.6. In particular, that section discusses

the main differences between workload scheduling in classical parallel environments (such as multiprocessors and clusters) and Grids. The results of this off-line scheduling model are then used in on-line scheduling algorithms as described in chapter 6. Section 2.7 at last highlights the importance of accurate Grid simulation environments allowing for realistic and repeatable experiments on Grids. We identify key features of existing Grid simulation frameworks, and compare them to the Grid simulation environment developed in the course of our research. This environment is thoroughly described later on in chapters 3 and 4.

2.2 Grid Middleware

The glue needed to tie the various building blocks together and to exploit the aggregate power of the resulting Grid is provided by Grid *middleware*. This middleware supports authentication, authorization, resource advertisement, resource management, resource allocation, monitoring and job scheduling operations and ideally provides standardized interfaces allowing for the construction of Grid user interfaces enabling transparent and intuitive access to the underlying set of distributed resources (see figure 2.1). These user interfaces typically generate job descriptions (capturing application characteristics and resource requirements) in some *job description language*, which are then passed on to the appropriate middleware components.

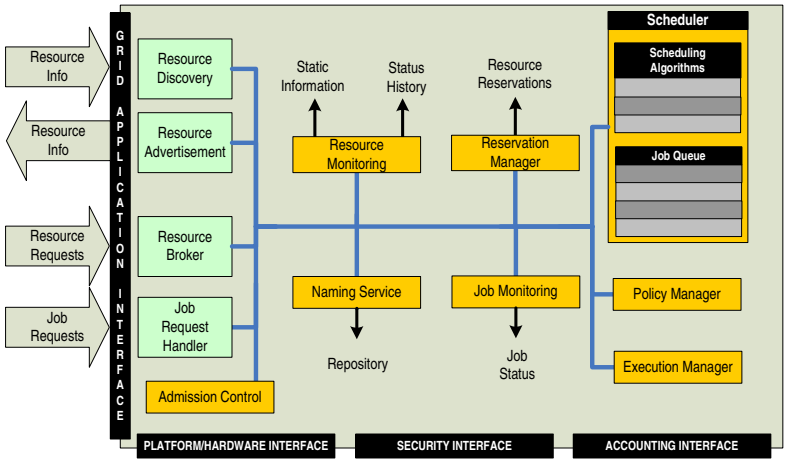


Figure 2.1: Grid Middleware: Components and Interfaces

Several Grid middleware solutions are in use today. Compute-intensive cycle-stealing applications (typically disguised as screensavers for desktop PCs) can be deployed using the Berkeley Open Infrastructure for Network Computing (BOINC [1]) or using custom software, as is the case with the World Community Grid [2].

Managing clusters as well as idle desktop machines running various operating systems with the intent of batch scheduling compute-intensive jobs on them is the goal of Condor [3], the distributed resource management system of the University of Wisconsin-Madison. Its functionality can be compared to that of Sun Grid Engine [4].

In the 1990's, an effort was initiated to enable several German supercomputer centers to provide secure and intuitive access to their heterogeneous set of computing resources. The result of this effort is known today as the UNICORE (Uniform Interface to Computing Resources [5]) Grid Environment.

During the same years work was started on a *metacomputing* toolkit called the Globus Toolkit [6], intended to realize the computing vision now called the Grid. This toolkit has evolved into the basis of a number of Grid projects of significant size. One notable example is the Large Hadron Collider Computing Grid (LCG [7]) which will process data generated by CERN's Large Hadron Collider and uses Grid middleware based upon the Globus Toolkit. While the LCG primarily deals with high energy physics (HEP) applications, its underlying Globus-based middleware has been re-used in the broader (i.e. addressing the entire scientific community) EGEE service Grid. The specific applications geared towards high energy physics as deployed in the LCG can be seen as one of the services offered by the EGEE Grid.

Over the past years, a lot of effort has gone into formalizing and standardizing Grid protocols and architectures within the Global Grid Forum. More specifically, a set of *services* and their corresponding interfaces have been identified in areas such as resource management, resource virtualization, data management and security. This has led to the definition of the Open Grid Services Architecture (OGSA [8]).

The latest incarnation of the Globus toolkit is arguably the most exhaustive offering of OGSA-compliant Grid services available today.

Important services assist in resource status querying, selection and co-allocation for job execution as well as in the partitioning of available resources into various Virtual Private Grids. Resource demands originate from the typical applications that need to be executed on the Grid. In collaborative data-processing environments (e.g. the Grid concept as needed by CERN), the need arises to transfer large amounts of data between distant Grid sites. Untimely delivery of such data may cause a degradation in application performance. Consequently, in this setup, not only computing resources are of importance; the nature and careful exploitation of network resources is critical as well.

In this context, the relevance of using optical technologies to interconnect dispersed Grid sites has vastly increased [9]. The use of optical networks as the Grid resource interconnection technology of choice is discussed in section 2.4.

As shown in figure 2.1, resource monitoring also forms a major Grid middle-

ware aspect. Monitoring data reveal statistics on Grid resource usage efficiency and can be fed back to Grid schedulers in order to improve resource allocation decisions. The importance of resource monitoring has spawned numerous projects attempting to implement resource monitoring functionality. An overview of some well-known projects in this area is presented in section 2.3. In that same section, the rationale behind the development of our own resource monitoring architecture - developed in parallel with our research work - and its comparison to existing solutions are presented.

2.3 Grid Monitoring

In order for the Grid middleware to be able to perform informed resource allocations, it needs to have access to up-to-date resource state data. These data are typically gathered by a *monitoring* service. Conversely, monitoring data concerning application characteristics, data access patterns and resource usage efficiency in an operational Grid is not only useful as input into the deployed resource allocation algorithms, but can be used to improve these algorithms.

The governing body for Grid standardization [10], the Global Grid Forum, has recognized the importance of such Grid monitoring systems. It can be argued that any successful Grid monitoring system is at least required to be scalable (due to the large size of Grids), portable (due to the presence of different computing resources and operating systems), extensible (due to the variety of resource types connected to a Grid) and efficient (in order to minimize monitoring overhead). The Global Grid Forum has therefore launched the *Grid Monitoring Architecture* (GMA), a reference architecture for feasible Grid monitoring systems adhering to the aforementioned properties. The overall structure of a GMA compatible framework is depicted in figure 2.2.

Three major components can be identified in the GMA: producers, consumers and a directory service. The directory service stores the location and type of information provided by the different producers, while consumers typically query the directory to find out which producers can provide their needed event data (after which they contact the producers directly). Producers in turn can receive their event data from a variety of providers (software/hardware sensors, applications, whole monitoring systems, databases, etc.). The GMA does not specify the underlying data models or protocols that have to be used.

Multiple monitoring architectures for Grid-like systems have already been successfully deployed. Not all of them follow the guidelines set by the GMA (e.g. Condor's Hawkeye [11] which does not support a decentralized architecture), and some are geared towards monitoring one single resource type (e.g. Remos [12], focussing on network parameters).

The NetLogger toolkit [13, 14] allows for the monitoring of distributed appli-

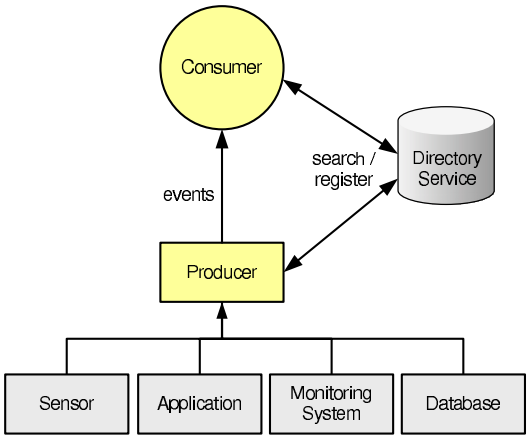


Figure 2.2: Grid Monitoring Architecture: Overview

cations, which must be modified and instrumented by the developers to generate suitable events. Currently, NetLogger uses a centralized data repository which does not scale well to Grid-scale environments.

The Network Weather Service [15] is capable of monitoring and predicting the performance of network and computational resources. Its statistical prediction capabilities have mainly been used to support dynamic schedulers with Quality of Service information. Again, however, some components in the architecture (e.g. the forecaster and name server) are completely centralized.

Nagios [16] offers a set of tools to monitor both resources and network services. Sensor data is provided by suitable plugins and can be published on a web page. Its deployment is limited to Unix-like operating systems.

One major GMA compliant Grid monitoring system is the European DataGrid's Relational Grid Monitoring Architecture R-GMA [17]. R-GMA offers a combined monitoring and information system using a Relational Database Management System as directory service and monitoring data repository (this approach offers the possibility to formulate complex queries on the monitored data i.e. it allows to locate monitoring components and retrieve the data they offer using standard SQL statements). The implementation is based on Java servlet technology (using the Tomcat servlet container), trading performance for portability and limited software dependencies. The complete EU DataGrid monitoring architecture also includes the Mercury [18] application progress tracer, which is a Grid-enabled version of the the GRM distributed monitor part of the GRM [19, 20] message-passing application instrumentation library used in traditional parallel environments.

Another Grid monitoring system is GridRM [21]. GridRM is an open source two-layer Grid monitoring framework, the upper layer being structured according

to the GMA. This upper layer connects the per-site monitoring systems (the lower layer) in a scalable way. Like R-GMA, GridRM makes use of Java and SQL to query data producers. Currently, GridRM's directory service (containing info on the location of the different resource status providers) has shown to be a possible bottleneck and single point of failure.

MDS2 is the Globus toolkit (version 2) Monitoring and Discovery Service, and although MDS development was started before the GMA was conceived, it can still be regarded as an implementation of the Grid Monitoring Architecture. MDS2 only supports latest-state queries (as opposed to retrieving resource state history), making it mandatory for the consumers to actively retrieve status information from the GRIS (the MDS2 component offering producer-like functionality). The later versions of the Globus toolkit (version 3 and up) have replaced MDS2 with a GMA compatible monitoring and information framework implemented using Java and web services technology, the so-called Web Services based Information Service [22].

JAMM (Java Agents for Monitoring and Management) [23] is a Java based monitoring architecture, based on the GMA. It offers automatically deployed sensor agents, but most of these agents are actually wrappers around Unix tools and can therefore not be deployed on other operating systems.

For an extensive comparison of Grid monitoring frameworks, both feature-wise and performance-wise, we refer to [24]. A detailed classification of the systems listed in this section based on their compliance with and level of implementation of the GMA is given in [25].

During the course of our research, an alternative GMA compatible Grid monitoring and information framework has been implemented. The underlying ideas were to implement a highly performant GMA compatible monitoring framework, supporting more advanced consumer and producer types such as long-term archiving consumers and real-time visualization consumers, as these features are not common, nor are they major focal points, among the alternatives listed above.

We have described the architectural details and implementation decisions concerning our Grid Monitoring Architecture in appendix A.

As far as our implementation's performance is concerned, a thorough comparison to the Globus MDS2 system and its successor has been carried out and described in [26, 27] and appendix A. Only the Globus monitoring and information framework was used as benchmark, as a comprehensive study [24] has already shown the Globus MDS2 system to outperform (i.e. MDS2 exhibits lower response times and better scalability) the other major GMA compatible frameworks listed here.

2.4 Optical Transport Networks

Technological advantages of optical transport networks (OTNs) include the ability to carry high-bit rate data over large distances, without excessive need for amplification and without performance-limiting effects such as cross-talk. It is this ability to reliably transport huge amounts of data in a timely fashion that has drawn widespread attention from the Grid community.

A major drawback in the design of optical network routers, however, is the difficulty of providing temporary storage (i.e. memory) for optical data. This implies that it is non-trivial to operate an optical network in a *packet switched* [28] mode of operation, as memory is needed to buffer packets while headers are being examined.

Another major mode of operation of optical transport networks involves the use of *lightpaths*, which provide an end-to-end bandwidth pipe, similar to a POTS circuit. The end-to-end lightpaths are established by concatenating a number of wavelengths on a fiber route between the endpoints. Intermediate optical cross-connects may perform wavelength conversions such that a lightpath need not be carried on the same wavelength along its entire route.

These *circuit switched* [29] optical networks establish guaranteed-bandwidth pipes, but in doing so exploit the network's capacity less efficiently when compared to the flexibility offered by packet-switching networks.

Because of this, and the implementation difficulties associated with packet switched optical networks, a third operational mode called *burst switching* [30] receives widespread attention. In a burst switching setup, data is sent in bursts, where a burst has a finite length and occupies a single wavelength. Unlike a packet, which contains control information in its header, all control information related to a data burst precedes it in a dedicated control burst. The time offset between the control burst and the actual data burst is of sufficient magnitude to allow intermediate routers to determine the preferred route for the burst and to allocate the required resources (wavelength/time window pairs). This construction reduces the need for optical storage buffers.

For the data-intensive scientific applications mentioned earlier, a long-lived set of dedicated wavelength paths between sites (thus, a circuit switched optical transport network connecting the Grid sites) forms a natural way of providing the necessary bandwidth in an adequate fashion. For instance, European research and education networks (such as Belnet [31]) are interconnected using multiple $10Gbps$ wavelengths by the GÉANT2 [32] network, while Abilene [33] and CA*Net4 [34] have a similar role in the USA and Canada, respectively. Lambda networking is heavily promoted by the Global Lambda Integrated Facility (GLIF [35, 36]) virtual organization. GLIF participants jointly make lambdas available as an integrated global facility for use by scientists and projects involved in data-intensive

scientific research.

The possibilities of a burst switched network, however, can open up the concept of Grid computing to other classes of applications, which may be highly interactive and operate on a variety of data sets - examples include multimedia editing and virtual reality immersion. For these classes of applications, it is infeasible and inefficient to set up and tear down lightpaths for every individual data transfer. In contrast, a job and the data it is to operate on can be combined into a burst, which can then be submitted onto the network without performing tedious lightpath set up and tear down operations. The resulting burst can then be routed to a suitable processing destination following an *anycast* scheme as proposed in [37].

While burst switching is a very promising technology, issues that remain to be solved satisfactorily include the routing and deflection of bursts in case of network congestion. The current lambda Grid deployments mainly use circuit switched optical networks to ensure site interconnections. In what follows, we dissect the network stack used in these interconnections.

In optical data transmissions, information is carried by propagating light through a suitable waveguide called an optical fiber. At the transmitter end, the light beam (mostly in the wavelength range of $1300-1550nm$) is injected into the fiber's core by a laser. At the receiving end the light hits a detector which emits an electrical signal. The highest transmission rates (order of $40Gbps$) can be obtained with single-mode fibers. These fibers feature a small core and only allow a single electromagnetic wave mode to propagate. Single-mode fibers require more expensive equipment to operate, however, than their multi-mode counterparts. While optical fibers clearly present an opportunity to transmit data at very high rates, their capacity has been further expanded by the advent of the wavelength division multiplexing (WDM) technology. As its name indicates, WDM allows to use multiple optical signals simultaneously on a single fiber by spacing the signals in the frequency domain. Two different classes of WDM technology are usually considered. The first class, called coarse WDM (CWDM), denotes those WDM transmission systems offering limited multiplexing capabilities (4, 8 or 16 wavelengths per fiber, coarsely spread over the frequency band). As the different signals in these systems can be widely separated, CWDM systems are cheaper than their DWDM counterparts. DWDM (dense WDM) systems can multiplex up to 160 different signals onto a single fiber. As these signals are placed very close together in the frequency band, DWDM systems need more expensive equipment to operate.

Because WDM has increased the available network bandwidth, the need arose to enable faster switching in the network. Initially, only point-to-point WDM systems were deployed. Nowadays, optically switched networks are made possible by the introduction of optical cross-connects (OXCs). These optical cross-connects can switch the signal arriving on a wavelength on an incoming link to the same wavelength on an outgoing link. This way, an end-to-end wavelength path utilizing

the same wavelength on each fiber segment can be set up. If the OXC is equipped with wavelength converters, the outgoing wavelength of the switched signal can be different from the incoming wavelength. This latter (more expensive) type of OXC allows the creation of so-called virtual wavelength paths, in which an end-to-end lightpath utilizes different wavelengths on its constituent fiber segments. Figure 2.3 shows the most important elements within such an OXC. Incoming fibers enter the OXC at the left; demultiplexers retrieve the individual wavelengths on each fiber. A space-switching matrix connects this incoming wavelength to an outgoing multiplexer. This wavelength is converted if necessary into a different wavelength and multiplexed onto an outgoing fiber. From figure 2.3, which

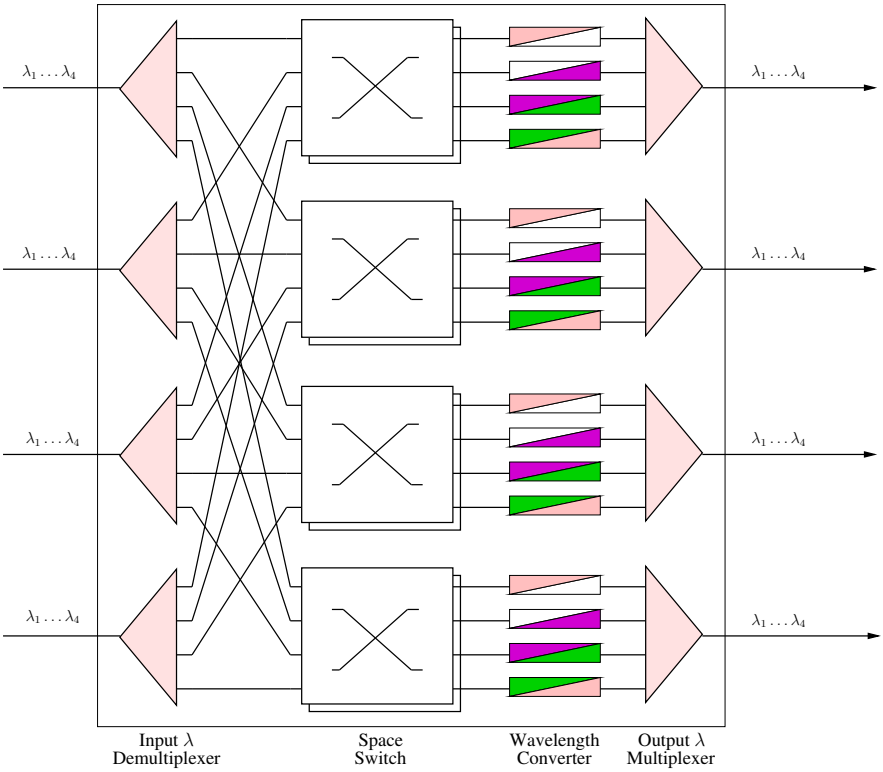


Figure 2.3: Optical Cross-Connect: Key Structural Components

represents an OXC with $F = 4$ in- and outgoing fibers each capable of carrying $N = 4$ wavelengths, it follows that the OXC's switching capabilities are determined by the switching matrix's complexity (and hence its blocking behavior) and the wavelength conversion opportunities. The OXC in figure 2.3 provides F $N \times N$ switching blocks, which is clearly more scalable yet less powerful than a single monolithic $NF \times NF$ switching block. When setting up (virtual) wave-

length paths in this work, we have assumed unlimited switching and wavelength conversion capabilities in each optical cross-connect, whilst observing wavelength capacity and flow constraints. Figure 2.4 shows how a switched optical network allows for the setup of end-to-end lightpaths.

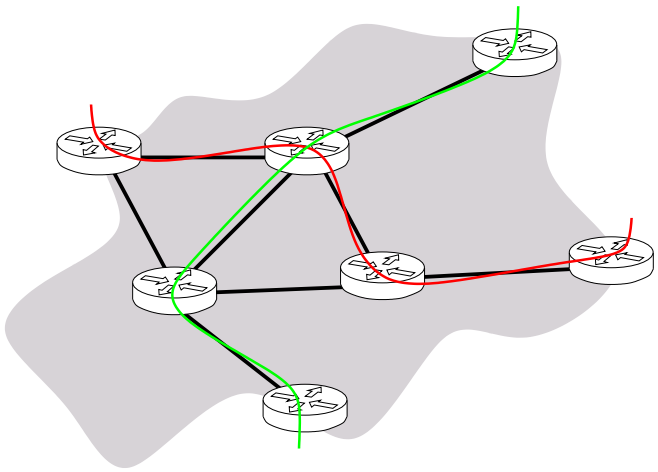


Figure 2.4: End-to-End Lightpaths in Switched All-Optical Network

In legacy transport networks, traffic carried on the fibers is structured using SDH/SONET, providing standardized data rates and multiplexing schemes for synchronous digital data transmission. Optimized for fixed-bit rate traffic (e.g. voice traffic), data is piggy-backed in these transport networks. As a result, the network stack for IP traffic can be quite complex - a sample protocol stack used to deliver IP traffic is depicted in figure 2.5(a). In this stack, ATM cells are mapped into SDH containers. In the ATM network, multiple routed protocols can be multiplexed within a single Virtual Circuit (VC) by using Logical Link Control (LLC) encapsulation. An example of such a routed protocol and the one shown in figure 2.5(a) is IP.

The bulk of the data carried, however, originates and/or ends up in an Ethernet network [38], the cost-effective dominant LAN solution. This has led to an increased interest in the Ethernet-over-WDM concept, in which Ethernet frames are mapped directly onto WDM wavelengths. The Ethernet-over-WDM approach allows to eliminate the overhead, complexities and expensive equipment involved with the legacy transport systems [38].

The transportation of Ethernet frames over a WDM system has already been standardized. For instance, the IEEE 802.3ae standard ($10GbE$) defines the 10GBASE-LX4 interface which maps a $10Gbps$ Ethernet signal onto 4 wavelengths of a (C)WDM fiber pair. Such $10Gbps$ Ethernet pipes are already used in e.g. the TeraGrid project to connect the various Grid sites. For that reason, we will fre-

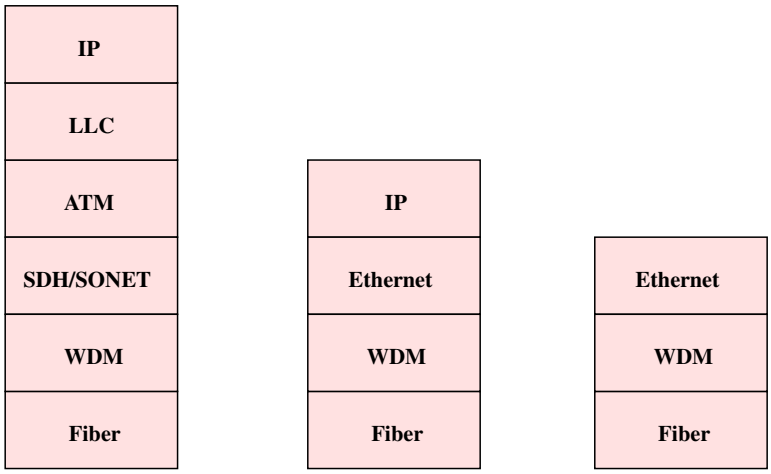


Figure 2.5: Data Traffic Protocol Stack: Legacy (a) vs. Ethernet-over-WDM, TCP/IP compatibility stack (b) and envisioned future stack (c)

quently use comparable parameters (i.e. 4 wavelengths per fiber, each wavelength supporting a $2.5Gbps$ data rate) throughout this work.

The resulting protocol stack for Ethernet-over-WDM is illustrated in figures 2.5(b) and 2.5(c). Stack 2.5(b) ensures compatibility with existing TCP/IP applications, while eliminating the IP layer and using Layer-2 protocols instead of TCP/UDP [38] as shown in figure 2.5(c) requires application modifications.

As far as the upper layers of the network stack are concerned, it is well known that standard TCP implementations can misbehave in high bandwidth-delay product (BDP) networks [39]. Consequently, there have been significant research efforts to either improve TCP implementations or to create alternative transport protocols for high BDP networks. While alternatives to TCP such as RBUDP [40], SABUL [41] and GTP [42] exist and can be deployed in high BDP networks, it has been shown that TCP itself can be scaled for high BDP networks [39] and that the problems associated with TCP originate in its implementation and are not intrinsic to TCP.

In the application layer, a new Grid-oriented file transfer protocol (based on FTP) allows to transfer data between different storage elements in the Grid. This GridFTP [43] protocol is geared towards reliable high-profile data movements using parallel and striped file transfers, and employs direct control over TCP window sizes to implement these features. Securing such file transfers is another major element in the GridFTP specification; to this end, GridFTP seamlessly integrates with the Grid Security Infrastructure (GSI [44]) and Kerberos [45].

Further enhancements to the GridFTP protocol (leading up to GridFTP version 2) are currently under investigation and include dynamic bandwidth management, bidirectional parallel transfers and transparent firewall / NAT traversal.

2.5 Optical Transport Network Dimensioning

As explained in the previous section, communication and data transfer between different Grid sites in lambda Grids occurs over an optical transport network (OTN). In currently deployed lambda Grids (e.g. GLIF [35] or TeraGrid [46]), circuit switching is the most widely used wavelength allocation method. In a circuit switched network, (virtual) wavelength paths are set up between edge nodes. These wavelength paths are carried across optical fibers connecting the optical cross connects in the network. As each wavelength path can carry a finite amount of data and the number of wavelengths that can be activated simultaneously is also finite, one should decide on the wavelength paths to be activated and their respective routing in the network before the network becomes operational. Each decision incurs a certain cost; typical cost components include the number of fiber ducts, the amount of fiber needed and the number of wavelengths activated on each fiber segment.

These decisions make up the OTN dimensioning problem aimed at obtaining network dimensions with expected network traffic in mind. Note that the use of wavelength paths ensures that OTN dimensioning suffers from a significant extra complexity when compared to standard network flow problems utilizing continuous bandwidth values. Indeed, in an OTN dimensioning problem bandwidth granularity is determined by each wavelength's capacity. In addition, for each wavelength path wavelength continuity or conversion constraints must be fulfilled.

While the scheduling of lambda requests in such a OTN has been addressed in [47] (emphasizing dynamic on-demand lightpath provisioning schemes), we are primarily interested in the establishment of long-lived pre-established wavelength paths. A lot of research has been dedicated to this OTN dimensioning problem featuring static traffic demands between node pairs [48–50]. Static network dimensioning starts from a given demand matrix, i.e. a matrix representation of the traffic demands between each pair of nodes in the network. In the case of optical circuit switched networks, the network dimensioning problem is called a Routing and Fiber and Wavelength Assignment (RFWA) problem [49, 50] for obvious reasons.

This type of problem can be modeled as a multicommodity network flow problem [48], where every commodity maps to a single source-destination pair of nodes in the network. These multicommodity network flow problems can be formulated as integer linear programs as follows.

Let $\mathcal{N}$ be a set of nodes and $\mathcal{E}$ be a set of directed edges connecting these nodes.

Furthermore, assume a demand matrix D (i.e. d_{ij} is the number of wavelength paths required to be set up from node i to node j) is given. Introducing binary variables $f_{\lambda kl}^{ij}$ (with $(k,l) \in \mathcal{E}$) denoting whether or not a wavelength path from node i to node j is being carried on the edge between nodes k and l on physical wavelength $\lambda \in \Lambda$, we have that

$$\forall i \in \mathcal{N}.\forall j \in \mathcal{N} \setminus \{i\}. \sum_{l:(i,l)\in\mathcal{E},\lambda\in\Lambda} f_{\lambda il}^{ij} = d_{ij} \tag{2.1}$$

$$\forall i \in \mathcal{N}.\forall j \in \mathcal{N} \setminus \{i\}. \sum_{k:(k,j)\in\mathcal{E},\lambda\in\Lambda} f_{\lambda kj}^{ij} = d_{ij} \tag{2.2}$$

$$\forall i \in \mathcal{N}.\forall j \in \mathcal{N} \setminus \{i\}.\forall k \in \mathcal{N} \setminus \{i,j\}. \sum_{m:(m,k)\in\mathcal{E},\lambda\in\Lambda} f_{\lambda mk}^{ij} = \sum_{n:(k,n)\in\mathcal{E},\lambda\in\Lambda} f_{\lambda kn}^{ij} \tag{2.3}$$

Equations 2.1, 2.2 and 2.3 deal with flow conservation in each flow's source, destination and intermediate router nodes, respectively. A typical cost function to be minimized in this type of problem is the aggregate number of wavelengths activated in the network, given by

$$\sum_{(i,j)\in\mathcal{N}^2,(k,l)\in\mathcal{E},\lambda\in\Lambda} f_{\lambda kl}^{ij} \tag{2.4}$$

In general, such a multicommodity flow problem is NP-complete, meaning that no algorithm is known to find an optimal solution for each such problem within a time horizon polynomial in the problem's dimensions. This is aggravated by the fact that the given integer linear program as formulated above suffers from a large amount of integer variables needed to discriminate between the various commodities (i.e. flows) and the available wavelengths on each fiber.

Note, however, that cost function 2.4 does not depend on e.g. the exact allocation of wavelengths to lightpaths, nor does it depend on the number of wavelength translations occurring in each node. For this class of optimization objectives, the integer linear program modeling the multifiber RFWA problem can be reduced in complexity by employing the concept of a *source routing* formulation as introduced in [51].

In such a formulation, variables $f_{\lambda kl}^{ij}$ are replaced by variables f_{kl}^{i}, denoting the (integer) number of wavelength paths originating at node i carried on the link between nodes k and l. These variables do not record each wavelength path's destination, nor do they assign wavelengths to each section of the path. As shown in [51] however, this does not prevent the correct modeling of the RFWA problem as an integer linear program, provided the optimization objective adheres to the constraints laid out above.

Using these replacement variables, wavelength path routing constraints 2.1,2.2 and 2.3 become

$$\forall i \in \mathcal{N}. \sum_{l:(i,l)\in\mathcal{E}} f_{il}^{i} = \sum_{j\in\mathcal{N}\setminus\{i\}} d_{ij} \tag{2.5}$$

$$\forall i \in \mathcal{N}. \forall j \in \mathcal{N} \setminus \{i\}. \sum_{k:(k,j)\in\mathcal{E}} f_{kj}^{i} = d_{ij} + \sum_{l:(j,l)\in\mathcal{E}} f_{jl}^{i} \tag{2.6}$$

Cost function 2.4 now reduces to

$$\sum_{i\in\mathcal{N},(k,l)\in\mathcal{E}} f_{kl}^{i} \tag{2.7}$$

Observe how this alternative set of variables has simplified the modeling of the OTN dimensioning problem from a static demand matrix.

Of course, in a lambda Grid setting such a given, static demand matrix between Grid site pairs is nonexistent. Instead, the resulting traffic in the Grid's interconnection network has its origins in the way workload is distributed and scheduled across the participating Grid sites. The approach used in our work to combine these scheduling decisions into the OTN dimensioning problem is detailed in chapter 5. In that same chapter, we have also extended this combined problem to take into account possible resource and network element failures. Techniques to improve this resulting combined scheduling and dimensioning problem's scalability are detailed in the next section.

2.6 Workload scheduling

In this section, we take a closer look at the problem of scheduling workload on the different resources in a Grid. In particular, we are interested in the off-line modeling of such a Grid workload scheduling problem.

In a Grid setting, it is of paramount importance that several Grid resources can be co-allocated to a single job. Consider a simple data processing job accessing a remote repository. For such a job, both the CPU time allocated to it as well as network bandwidth on the path to the repository determine its run time and progress. This indicates that different types of first-class resources must be handled (in this case, both computational and network resources), that multiple of these resources can be allocated to a single job, and that the allocations made for this job on these resources are interdependent. To illustrate this last statement, consider the case where available network bandwidth to the remote data repository is low. If this is the case, then we can increase the CPU share allocated to the job on its computational resource at will without reducing the resulting job's finishing time, as the job will be stalling and waiting for the remote data to arrive.

From this example, it follows that the modeling of an off-line Grid workload scheduling problem differs significantly from the related problem of workload scheduling in a cluster or multiprocessor environment. In the latter type of environment, the off-line workload scheduling is most often the problem of scheduling individual jobs on a set of processing elements (e.g. the nodes in the cluster or the processors in the multiprocessor machine). This off-line scheduling of jobs on resources of a single type (i.e. computational resources) has been studied extensively in literature [52–54]. In these works, the quality of on-line scheduling heuristics has been analyzed and compared extensively to the optimal solution to the off-line problem.

The off-line workload scheduling problems in this context are modeled as (integer) linear programs, and can be treated as special instances of Multi-Modal Resource Constrained Project Scheduling Problems (MMRCPSP) as described in [55–59]. In this class of problems, a workload with fixed a priori resource requirements (for different types of resources) is scheduled on a set of such resources. A standard MMRCPSP can be expressed as a linear program as follows:

Let $\mathcal{R}$ be a set of renewable resources, and assume that at each moment in time the capacity of resource k is fixed at R_k. Let there be a set of activities and assume each activity j can be processed in a number of different modes $\mathcal{M}_j$. Within mode m, activity j uses r_{jkm} resource units of resource k at all times and runs for a number of time units p_{jm}. Thus, choosing a mode of execution for activity j decides upon the resources used for this activity as well as the amount of resource units needed. For given inputs $R_k, \mathcal{M}_j, r_{jkm}, p_{jm}$ a valid non-preemptive project schedule then consists of

- the assignment of a mode of execution to each activity j
- the scheduling (i.e. decision on the starting time) of each activity j
- the enforcement of resource capacity constraints
- the minimization of some metric (e.g. makespan or average completion time) while making the above decisions

Introducing binary variables s_{jmt} equalling 1 if and only if activity j is scheduled to start in mode m at time $t \in \mathcal{T}$, the unique mode assignment and scheduling of the activities (which combines with the resource capacity constraints) can be expressed as

$$\forall j \in \mathcal{J}. \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} s_{jmt} = 1 \tag{2.8}$$

$$\forall k \in \mathcal{R}. \forall t \in \mathcal{T}. \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t' \in \{t - p_{jm} + 1, \ldots, t\}} s_{jmt'} r_{jkm} \leq R_k \tag{2.9}$$

To minimize e.g. the schedule's average completion time, one would have to minimize

$$\sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t \in \mathcal{T}} s_{jmt}(t + p_{jm}) \tag{2.10}$$

It is easy to see that this kind of formulation can be used to model a workload scheduling problem with rigid resource requirements by simply replacing the terms "activity" and "mode" by "job" and "allocated resource set", respectively.

In general, however, solving these integer linear programs is an NP-complete problem. In practical terms, this means that the process of identifying an optimal solution to such a problem cannot be bounded (in time) polynomially in terms of the problem's dimensions (i.e. the number of jobs, the number of resources and the number of discrete time instants investigated). In addition, the resource requirements imposed by the jobs in the above linear program are rigid, that is, they are fixed and known in advance, are not related to each other and thus cannot model interdependent resource allocations nor can they model the time-sharing of a single resource over multiple jobs.

From these restrictions, it follows that the MMRCPSP approach cannot capture the complexity of the Grid workload scheduling problem (featuring a large number of jobs and resources and interdependent resource co-allocations) in its entirety.

Therefore, instead of starting with an off-line model to the scheduling problem and then proposing on-line heuristics to generate "suitable" (when compared to the off-line solution) workload schedules, many authors have used simulation tools to study the Grid-specific aspects of the workload scheduling problem. Ranganathan et al. [60] have studied independent CPU-allocation and data set replication through simulation. In a similar way, replica optimization is the main topic of the research described in [61]. That work focuses on the location of data sets, but does not address network resource allocations that may be needed to access the data within a deterministic time frame.

In [62], the effects of data needing to be transferred across the network is not expressed using resource allocations but is condensed into a single *overhead* parameter describing the slowdown of a job relative to the situation where the data is locally available. A similar parameter (called *slowdown factor*) appears in [63, 64], where queueing model analysis is performed to deduce schedule quality in a purely space-shared multicomputer system.

Market-driven and incentive-based parameter sweep application scheduling on computational resources has been studied extensively by Buyya et al. [65], where resource selection policies constrained by the notions of *budgets* and *deadlines* are investigated. Of course, these constraints influence the amount of work that can be performed by the Grid, while we are interested in effective scheduling of workload on a correctly dimensioned Grid.

The effects of co-allocating CPU and network resources to a single job have

been studied in [66]. Grid sites are connected through a VPN in which fine-grained bandwidth pipes can be set up. In reality, it is difficult to imagine a scenario in which such pipes with guarantees concerning delay, jitter and bandwidth availability can be set up over e.g. the Internet. In contrast, the use of optical transport networks does offer the reality of high-capacity bandwidth pipes between the various Grid sites and is therefore a focal point in our work.

To cope with the scalability issue mentioned above, divisible load theory (DLT) [67–69] has been used successfully in recent years in modeling steady state off-line Grid scheduling problems. Scheduling decisions concerning both computational and network resources (not necessarily optical) are derived from a scalable linear program. The improved scalability is obtained by only considering steady state operational scenarios (in contrast to dealing with discrete time instants) and divisible workload (in contrast to discrete individual jobs). The net result is that scheduling constraints in the linear program can now be expressed using real-valued variables (instead of integer variables), and the affected portions of the off-line scheduling linear program can now be adapted as follows.

Assume the steady state workload arriving in the Grid for resource type r is α_r. This workload needs to be distributed among all resources k of the relevant resource type. Assuming α_r^k is the amount of workload distributed to resource k of type r per time unit, equations 2.8 and 2.9 now become

$$\forall r. \sum_{k \in \mathcal{R}_r} \alpha_r^k = \alpha_r \tag{2.11}$$

$$\forall r. \forall k \in \mathcal{R}_r. \alpha_r^k \leq R_k \tag{2.12}$$

The assumption of steady state also requires a new optimization objective; a typical example of such an objective is to balance the load across all resources and thus to minimize e.g.

$$\max_{r,k \in \mathcal{R}_r} \frac{\alpha_r^k}{R_k} \tag{2.13}$$

Clearly, the assumptions on steady state and divisible workload have enabled a significant reduction in the linear program's complexity. An off-line scheduling problem based on this approach and involving both computational and network resources has been investigated in [69]. In that work, however, network resource allocations are in fact fixed-bandwidth TCP connections supplemented by a somewhat artificial maximum number of TCP connections allowed per network element. In contrast, in this book we primarily focus on lambda Grids. In those Grids, data is transferred over (virtual) wavelength paths in an optical transport network (OTN). These wavelength paths directly map onto the concept of a finite network resource, effectively replacing the artificial network constraints in the scheduling model of [69] by the required granularity and wavelength path continuity or conversion constraints. As scheduling workload on a remote site incurs

network traffic, we focus on the combined off-line scheduling and dimensioning problem (see also section 2.5) in this book rather than studying the off-line Grid workload scheduling problem by itself. In chapter 6, we derive and evaluate several on-line workload scheduling algorithms based on the solution to the combined off-line scheduling and dimensioning problem.

2.7 Grid Simulation

Because of the size of Grids (related to the number of resources involved), it is often impractical to build, operate and monitor Grid testbeds on a realistic scale. Furthermore, Grid resource management has stringent requirements not commonly found in more classical cluster setups, as resource allocations can be interdependent, and their respective sizes can be part of the scheduling problem rather than being given (see also section 2.6).

These non-trivial requirements make it inherently difficult to evaluate Grid resource management and scheduling policies using (tractable) analytical methods (e.g. queueing network analysis). Therefore, simulation is an important aid in the evaluation of Grid behaviour and Grid resource management policies. In addition, simulators allow for repeatable experiments under controlled circumstances.

This section aims to provide a detailed overview of existing Grid simulation tools, their main focus and, conversely, their possible limitations. In turn, we discuss the Grid simulators Bricks, MicroGrid, SimGrid, GridSim, OptorSim and ChicagoSim.

The Bricks Simulator [70, 71] focusses on client/server interaction in global high performance computing systems. It allows for a single centralized scheduling strategy, which does not scale well to large Grid systems and does not support the notion of multiple (competing) schedulers.

MicroGrid [72, 73] is an emulator modeled after Globus, allowing for the execution of Globus-enabled applications on a virtual Grid system. Research into the area of Grid scheduling algorithms can be cumbersome with this kind of approach, since it requires the construction of an actual application to test. Furthermore, the applications are run and simulated in real-time, contrasting with the approach used in discrete-event simulators.

SimGrid [74] is designed to simulate task scheduling (centralized or distributed) on Grids. Version 1 of SimGrid can be regarded as a low-level toolkit (which interfaces to the C programming language) from which domain-specific simulators can be built. The second version of SimGrid is dubbed MetaSimGrid [75] and is essentially a simulator built upon this toolkit to enable the construction of simulated scenarios featuring multiple schedulers (as C programs). Models for network links as well as for TCP connections are present in SimGrid. This validated TCP implementation allows for smaller simulation times when compared to the packet-level

TCP simulation performed by network simulators. Of course, simulations using other transport protocols that are not readily available in SimGrid require that these protocols are implemented first, whereas using a network simulator ensures easy access to a wide range of protocols. The simulated application consists of several tasks, organized into a Directed Acyclic Graph (DAG). MetaSimGrid is focussed on scheduling this application type in a master-slave environment.

GridSim [76, 77] is a discrete-event Grid simulator based on JavaSim [78] (which itself has in the meantime been superseded by JSIM [79], featuring a Tcl/Java dual-language design and interface). This simulator allows to model a distributed set of Grid schedulers, and specifically focusses on market-driven economic resource models. While its computational resource models are flexible and highly configurable, simulation of Grid site interconnections and underlying network dynamics is not as thorough.

OptorSim [80, 81] is a Java [82] based Grid simulator built with the explicit goal of evaluating the performance of data access and replica placement optimization algorithms in a Grid. It has been conceived within the EU DataGrid project [83], and this is reflected in its architecture. OptorSim includes an economic model, using a peer-to-peer auction protocol that optimizes both the selection of replicas for running jobs and the dynamic creation of replicas in Grid sites using a file revenue prediction function. While OptorSim takes network bandwidth into account when transferring data between replica sites, it does not actually simulate any existing network or transport protocols.

The Chicago Simulator [84, 85] is a simulation framework built on top of Parsec [86] for studying scheduling and replication strategies in Grids. It provides a "condensed" view of the Grid's interconnecting network, as the bandwidth available to each Grid site's gateway is the major parameter in the network model it uses.

The Grid simulation environment we developed is called NSGrid, and is treated in chapters 3 and 4. This simulation environment encompasses the best features found in the above Grid simulators. Amongst others, NSGrid provides a dual layer mixed Tcl/C++ interface and network packet-level simulation by means of being constructed on top of the existing and widely used network simulator ns-2 [87]. This latter feature gives NSGrid an edge when it comes to network resource and transport protocol modeling. In addition, NSGrid not only provides models for the traditional Grid resources (computation, storage, network), but also provides models for important Grid middleware components such as schedulers and directory services and the communication between these components.

2.8 Conclusions

In this chapter, we have identified major concepts and technologies supporting the notion of lambda Grids. For each of these concepts, we have highlighted relevant research activities and results and we have indicated how our own research contributions presented in the following chapters build upon the concepts described in this chapter.

We focussed on the necessary Grid middleware and in particular the scheduling and resource monitoring components. We argued in favor of the use of optical transport networks in Grids (hence the term lambda Grid) and explained their operation. Furthermore, we gave an overview of modeling techniques used in optical network dimensioning and workload scheduling problems and pointed out the necessary changes to be made for these models to remain useful in a Grid context.

Lastly, we emphasized the importance of an accurate Grid simulation framework and discussed why we believe the development of a new Grid simulation framework called NSGrid was preferable.

References

[1] *Berkeley Open Infrastructure for Network Computing*. http://boinc.berkeley.edu/.

[2] G.M. Morris, D.S. Goodsell, R.S. Halliday, R. Huey, W.E. Hart, R.K. Belew, and A.J.W. Olson. *Virtual Screening using Grid Computing: the Screensaver Project*. Journal of Computational Chemistry, 19:1639–1662, 1998.

[3] Douglas Thain, Todd Tannenbaum, and Miron Livny. *Condor and the Grid in Grid Computing: Making The Global Infrastructure a Reality*. John Wiley, 2003.

[4] *Grid Engine Project Home*. http://gridengine.sunsource.net.

[5] A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder. *UNICORE – from Project Results to Production Grids*. preprint, 2005.

[6] *The Globus Alliance*. http://www.globus.org/.

[7] *LHC Computing Grid project*. http://lcg.web.cern.ch/LCG.

[8] I. Foster and al. *The Open Grid Services Architecture, Version 1.0*. draft-ggf-OGSA-spec-019 http://forge.gridforum.org/projects/ogsa-wg.

[9] D. Simeonidou, R. Nejabati, B. St. Arnaud, M. Beck, P. Clarke, D. B. Hoang, D. Hutchison, G. Karmous-Edwards, T. Lavian, J. Leigh, J. Mambretti, V. Sander, J. Strand, and F. Travostino. *Optical Network Infrastructure for Grid*. Global Grid Forum Informational Document, 2004.

[10] *Global Grid Forum*. http://www.gridforum.org/.

[11] *HawkEye: A Monitoring and Management Tool for Distributed Systems*. http://www.cs.wisc.edu/condor/hawkeye/.

[12] Bruce Lowekamp, Nancy Miller, Thomas Gross, Peter Steenkiste, Jaspal Subhlok, and Dean Sutherland. *A resource query interface for network-aware applications*. Cluster Computing, 2(2):139–151, 1999.

[13] B.L. Tierney and D. Gunter. *NetLogger: a Toolkit for Distributed System Performance Tuning and Debugging*. LBNL Technical Report LBNL-51276, 1997.

[14] D. Gunter, B.L. Tierney, C.E. Tull, and V. Virmani. *On-Demand Grid Application Tuning and Debugging with the NetLogger Activation*. In Proceedings of the 4th International Workshop on Grid Computing (Grid2003), pages 76–83, 2003.

[15] R. Wolski, N. Spring, and Jim Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Future Generation Computer Systems, 15(5-6):757–768, 1999.

[16] A. Douitsis. *Service Level Monitoring with Nagios*. TERENA Networking Conference 2005, 2005.

[17] A. Cooke, A.Gray, L. Ma, W. Nutt, J. Magowan, P. Taylor, R. Byrom, L. Field, S. Hicks, and J. Leake et al. *R-GMA: An Information Integration System for Grid Monitoring*. In Proc. of the 11th International Conference on Cooperative Information Systems, pages 462–481, 2003.

[18] N. Podhorszki, Z. Balaton, and G. Gombs. *Monitoring Message-Passing Parallel Applications in the Grid with GRM and Mercury Monitor*. In Proceedings of the 2nd European Across Grids Conference (AxGrids 2004), pages 179–181, 2004.

[19] N. Podhorszki and P. Kacsuk. *Design and implementation of a distributed monitor for semi-on-line monitoring of VisualMP applications*. In Proceedings of the Third Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS), pages 23–32, 2000.

[20] Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajdae. *From cluster monitoring to grid monitoring based on GRM*. In Proceedings of the Seventh International Euro-Par Conference, LNCS vol. 150, pages 874–881, 2001.

[21] M.A. Baker and G. Smith. *GridRM: A Resource Monitoring Architecture for the Grid*. In Springer-Verlag, editor, Proc. of the 3rd International Workshop on Grid Computing, pages 268–273, 2002.

[22] *WS Information Services website*. `http://www-unix.globus.org/toolkit/docs/3.2/infosvcs/ws/key/index.html`.

[23] Brian Tierney, Brian Crowley, Dan Gunter, Mason Holding, Jason Lee, and Mary Thompson. *A Monitoring Sensor Management System for Grid Environments*. In Proc. of High Performance Distributed Computing' 00, pages 97–104, 2000.

[24] X. Zhang, J.L. Freschl, and J. Schopf. *A Performance Study Of Monitoring and Information Services for Distributed Systems*. In Proc. of the 12th

IEEE International Symposium on High-Performance Distributed Computing, pages 270–282, 2003.

[25] Serafeim Zanikolas and Rizos Sakellariou. *A taxonomy of grid monitoring systems*. Future Generation Computer systems, 21:163–188, 2005.

[26] B. Volckaert, P. Thysebaert, M. De Leenheer, F. De Turck, B. Dhoedt, and P. Demeester. *A scalable and performant grid monitoring and information framework*. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, volume 1, pages 224–230, 2005.

[27] S. De Smet, P. Thysebaert, B. Volckaert, M. De Leenheer, D. De Winter, F. De Turck, B. Dhoedt, and P. Demeester. *A Performance Oriented Grid Monitoring Architecture*. In Proceedings of the 2nd IEEE Workshop on End-to-End Monitoring Techniques and Services (E2EMON), pages 23–28, 2004.

[28] M.J. O'Mahony, D. Simeonidou, D.K. Hunter, and A. Tzanakaki. *The application of optical packet switching in future communication networks*. IEEE Communications Magazine, 39:128–135, 2001.

[29] B. Mukherjee. *Optical WDM Networks*. Springer, 2006.

[30] C. Qiao and M. Yoo. *Choices, Features, and Issues in Optical Burst Switching*. Optical Networks Magazine, 1:36–44, 2000.

[31] *Belnet: A network of knowledge*. `http://www.belnet.be`.

[32] *GÉANT2*. `http://www.geant2.net`.

[33] *Abilene Backbone Network*. `http://abilene.internet2.edu/`.

[34] *CA*net 4*. `http://www.canarie.ca/canet4/`.

[35] *GLIF: Global Lambda Integrated Facility*. `http://www.glif.is`.

[36] F. Dijkstra and C. de Laat. *Optical Exchanges*. In First Workshop on Networks for Grid Applications (GridNets) - on CD-ROM, 2004.

[37] M. De Leenheer, F. Farahmand, P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, and J. Jue. *Anycast routing in optical burst switched grid networks*. In Proceedings of the 31st European Conference on Optical Communications (ECOC2005), pages 699–702, 2005.

[38] M.A. Ali, A. Hadjiantonis, H. Chamas, W. Bjorkman, and S. Elby. *On the Vision of Implementing a Truly Native Ethernet-Based Global Multi-Service Architecture*. In The 25th IEEE Conference on Computer Communications (IEEE INFOCOM) - on CD-ROM, 2006.

[39] A. Antony, J. Blom, C. de Laat, and J. Lee. *Exploring practical limitations of TCP over transatlantic networks*. Future Generation Computer Systems, Special issue: High-speed networks and services for data-intensive grids: The DataTAG project, 21:489–499, 2005.

[40] E. He, J. Leigh, O. Yu, and T.A. DeFanti. *Reliable Blast UDP : Predictable High Performance Bulk Data Transfer*. In The 4th IEEE International Conference on Cluster Computing, pages 317–324, 2002.

[41] Y. Gu, X. Hong, M. Mazzucco, and R.L. Grossman. *SABUL: A High Performance Data Transfer Protocol*. Journal of Grid Computing, 1:377–386, 2003.

[42] R. Wu and A. Chien. *GTP: Group Transport Protocol for Lambda-Grids*. In The 4th IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 228–238, 2004.

[43] W. Allcock. *GridFTP Protocol Specification*. Global Grid Forum Recommendation GFD.20, 2003.

[44] *Overview of the Grid Security Infrastructure (GSI)*. http://www.globus.org/security/overview.html.

[45] B.C. Neuman and T. Ts'o. *Kerberos: An Authentication Service for Computer Networks*. 32:1994, 33-38.

[46] *The TeraGrid project*. `http://www.teragrid.org/`.

[47] U. Farooq, S. Majumdar, and E. Parsons. *Dynamic Scheduling of Lightpaths in Lambda Grids*. In 2nd IEEE/Create-Net International Workshop on Networks for Grid Applications (GRIDNETS 2005), pages 1463–1472, 2005.

[48] D. Coudert and H. Rivano. *Lightpath Assignment for Multifibers WDM Networks with Wavelength Translators*. In Proceedings of IEEE Globecom'02, volume 3, pages 2686–2690, 2002.

[49] N. Wauters and P. Demeester. *Design of the Optical Path Layer in Multiwavelength Cross-Connected Networks*. IEEE Journal on Selected Areas in Communications, 14:881–892, 1996.

[50] D. Banerjee and B. Mukherjee. *Wavelength-Routed Optical Networks: Linear Formulation, Resource Budgeting Tradeoffs, and a Reconfiguration Study*. IEEE/ACM Transactions on Networking, 8:598–607, 2000.

[51] M. Tornatore, G. Maier, and A. Pattavina. *WDM network optimization by ILP based on source formulation*. In Proceedings of IEEE Infocom'02, volume 3, pages 1813–1821, 2002.

[52] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein. *Scheduling to minimize Average Completion Time: Off-line and On-line Approximation Algorithms*. Mathematics of Operations Research, 22(3):513–544, 1997.

[53] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong. *Theory and Practice in Parallel Job Scheduling*, pages 1–34. Springer Verlag, 1997.

[54] J. Sgall. *On-Line Scheduling - A Survey*. Lecture Notes in Computer Science, 1442:196–231, 1998.

[55] R. Kolisch and R. Padman. *An Integrated Survey of Project Scheduling*. OMEGA International Journal of Management Science, 29(3):249–272, 2001.

[56] N.P. Dhavale, S. Verma, and A. Bagchi. *Scheduling Partially Ordered Jobs Under Resource Constraints To Optimize Non-Regular Performance Measures*. IIMA Working Papers 2003-07-03, Indian Institute of Management Ahmedabad, Research and Publication Department, July 2003. available at http://ideas.repec.org/p/iim/iimawp/2003-07-03.html.

[57] A. Mingozzi and V. Maniezzo. *An Exact Algorithm for the Resource Constrained Project Scheduling Problem Based on a New Marthematical Formulation*. Management Science, 44:714–729, 1998.

[58] R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz. *Solving Project Scheduling Problems by Minimum Cut Computations*. Management Science, 49:330–350, 2003.

[59] P. Brucker. *Complex Scheduling Problems*. Osnabrücker Schriften zur Mathematik, Reihe P, No. 214, 1999.

[60] K. Ranganathan and I. Foster. *Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids*. Journal of Grid Computing, 1(1):53–62, 2003.

[61] D.G. Cameron, R. Carvajal-Schiaffino, A.P. Millar, C. Nicholson, K. Stockinger, and F. Zini. *Evaluating Scheduling and Replica Optimisation Strategies in OptorSim*. In 4th International Workshop on Grid Computing (Grid2003), pages 52–59, 2003.

[62] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. *Enhanced Algorithms for Multi-Site Scheduling*. In 3rd International Workshop on Grid Computing (Grid2002), pages 219–231, 2002.

[63] A.I.D. Bucur and D.H.J. Epema. *An Evaluation of Processor Co-Allocation for Different System Configurations and Job Structures*. In 14th IEEE Symposium on Computer Architecture and High Performance Computing, pages 195–203, 2002.

[64] A.I.D. Bucur and D.H.J. Epema. *The Influence of the Structure and Sizes of Jobs on the Performance of Co-Allocation*. In 6th Workshop on Job Scheduling Strategies for Parallel Processing, pages 154–173, 2000.

[65] R. Buyya, M. Murshed, and D. Abramson. *A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids*. In The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02) - on CD-ROM, 2002.

[66] P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, and P. Demeester. *Network Aspects of Grid Scheduling Algorithms*. In 17th International Conference on Parallel and Distributed Computing Systems (PDCS'04), pages 91–97, 2004.

[67] J.T. Hung, H.J. Kim, and T.G. Robertazzi. *Scalable Scheduling in Parallel Processors*. In 36th Annual Conference on Information Sciences and Systems (CISS'02), 2002.

[68] D. Yu and T.G. Robertazzi. *Divisible Load Scheduling for Grid Computing*. In 16th International Conference on Parallel and Distributed Computing Systems (PDCS'03), 2003.

[69] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. *A Realistic Network/Application Model for Scheduling Divisible Loads on Large-Scale Platforms*. In 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), page 48b, 2005.

[70] Atsuko Takefusa, Satoshi Matsuoka, Henri Casanova, and Francine Berman. *A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid*. In HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), pages 406–415, 2001.

[71] A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita. *Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications*. In Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), pages 34–47, 2003.

[72] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, Kenjiro Taura, and Andrew A. Chien. *The MicroGrid: a Scientific Tool for Modeling Computational Grids*. In Proc. of Supercomputing '00 - on CD-ROM, 2000.

[73] Xin Liu, Huaxia Xia, and Andrew Chien. *Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics*. Journal of Grid Computing, 2:141–161, 2004.

[74] Arnaud Legrand, Loris Marchal, and Henri Casanova. *Scheduling Distributed Applications: the SimGrid Simulation Framework*. In CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid, pages 138–145, 2003.

[75] J. Lerouge and A. Legrand. *MetaSimGrid : Towards realistic scheduling simulation of distributed applications*. ENS-LIP Research Report 2002-28, 2002.

[76] R. Buyya and M. Murshed. *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*. The Journal of Concurrency and Computation: Practice and Experience (CCPE), 14:1175–1220, May 2002.

[77] Anthony Sulistio, Gokul Poduvaly, Rajkumar Buyya, and Chen-Khong Tham. *Constructing A Grid Simulation with Differentiated Network Service Using GridSim*. In Proc. of the 6th International Conference on Internet Computing (ICOMP'05), pages 437–444, 2005.

[78] Hung-Ying Tyan and Chao-Ju Hou. *JavaSim: A component-based compositional network simulation environment*. In Proceedings of Western Simulation Multiconference - Communication Networks And Distributed Systems Modeling And Simulation, 2001.

[79] John A. Miller, Andrew F. Seila, and Xuewei Xiang. *The JSIM Web-Based Simulation Environment*. Future Generation Computer Systems (FGCS), Special Issue on Web-Based Modeling and Simulation, 17:119–133, 2000.

[80] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. *Simulation of Dynamic Grid Replication Strategies in OptorSim*. In GRID '02: Proceedings of the Third International Workshop on Grid Computing, pages 46–57, 2002.

[81] David G. Cameron, Ruben Carvajal-Schiaffino, A. Paul Millar, Caitriana Nicholson, Kurt Stockinger, and Floriano Zini. *Evaluating Scheduling and Replica Optimisation Strategies in OptorSim*. In 4th International Workshop on Grid Computing (Grid2003), pages 52–59, 2003.

[82] *Java*. http://java.sun.com/.

[83] *The DataGrid Project*. http://eu-datagrid.web.cern.ch/eu-datagrid/.

[84] K. Ranganathan and I. Foster. *Identifying Dynamic Replication Strategies for a High Performance Data Grid*. In Proc. of the International Grid Computing Workshop, pages 75–86, 2001.

[85] K. Ranganathan and I. Foster. *Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications*. In 11th Int. Symposium of High Performance Distributed Computing (HPDC), pages 352–358, 2002.

[86] *Parsec : Parallel Simulation Environment for Complex Systems*. http://pcl.cs.ucla.edu/projects/parsec.

[87] *The Network Simulator - NS2*. http://www.isi.edu/nsnam/ns.

3

NSGrid Grid Simulation Environment

3.1 NSGrid Rationale

As stated in section 2.7, simulation tools are a vital aid in assessing the quality of Grid scheduling and resource allocation policies, as the size of Grids (i.e. the number of resources, jobs and users involved) makes it very hard to build, operate and monitor Grid test beds on a realistic scale. In that same section, an overview was given of some well-known Grid simulation tools.

During the course of our research, significant effort was put into the development of a Grid simulation environment, combining the best properties of these existing simulation tools, and improving upon these solutions where appropriate. The resulting simulation environment, called *NSGrid* [1], offers the following major features:

- detailed Grid resource models, in particular models for computational resources, storage resources, data repositories and network elements
- generic and flexible Grid application model
- detailed Grid middleware models: software components and their interaction
- possibility of accurate packet-level simulation
- easy Grid simulation through high-level scripting language

- multiple scheduling and resource allocation algorithms built-in; new algorithms can easily be plugged in.

In this chapter, we give an overview of NSGrid's architecture, the simulation models it supports and its mode of operation. In the next chapter we showcase the use of NSGrid in two different application domains: the influence of network-awareness in Grid scheduling, and the partitioning of resources over different VOs or service classes.

3.2 NSGrid Architecture

The NSGrid simulation environment[1] is built around (and inherits its name from) ns-2 [2], the widespread network simulator. While not being the most scalable and efficient simulation kernel (in contrast to e.g. DaSSF [3, 4], ns-2 is a sequential simulation kernel) and being rather monolithic (as opposed to e.g. the OMNeT++ [5, 6] modular simulation environment), we chose ns-2 because it currently offers the most extensive range of network resource and transport protocol models of all publicly available (network) simulation environments today. The availability of this extensive array of models is partly due to ns-2 being widely accepted in an active research community and the exchange of such models within this community.

The relationship between ns-2 and NSGrid can be summarized as follows. Ns-

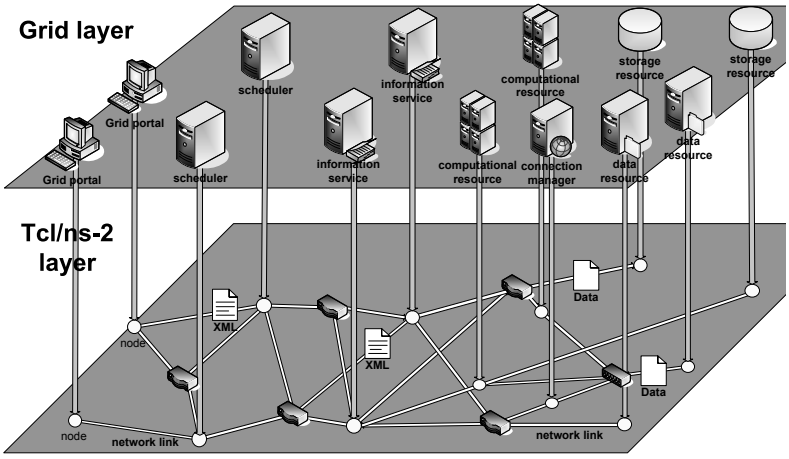


Figure 3.1: NSGrid layered architecture and relationship to ns-2

[1] NSGrid has been co-developed with Bruno Volckaert. His PhD thesis, titled "Architecturen en algoritmen voor netwerk- en dienstbewust Grid-resourcebeheer" contains a more detailed chapter on NSGrid's internals as well as its application in studying scheduling and VPG partitioning algorithms in Service Grids

2 is a discrete-event simulator and comes in a layered design. More accurately, two layers can be distinguished. On the one hand, we have the resource layer. In this layer, resource (in ns-2 these are mostly network related elements such as links, nodes and queues) behavioral models are implemented in C++, resulting in fast, compiled code. On the other hand, we have the simulation layer. In this layer, simulations are conceived as Tcl scripts. Resources for which models exist in the resource layer are exported to this simulation layer in which they can then be manipulated and configured. Special Tcl-C++ glue code is present to allow arguments and parameters, specified in a Tcl simulation driver script, to be parsed by the resource layer. Conversely, glue code also allows the resource models to return data to the driver script and to manipulate simulation layer objects, such as the simulation environment itself or the events that drive it.

Orthogonal to the resource/simulation layers, is the ns-2/NSGrid layering shown in figure 3.1. NSGrid adds models for common physical Grid resources such as computational resources, storage resources or read-only data repositories and replicas. These Grid resources, of course, do not float around freely. Instead, each Grid resource is located at a Grid site, and resources within a single site as well as the set of Grid sites are connected through a network. When modeling a Grid using NSGrid, this interconnecting network is created using ns-2 network objects such as nodes and links. Each NSGrid object (i.e. Grid resource) can then be attached to one of these nodes. The implied semantics of such an association are that all traffic destined for the NSGrid object involved (be it middleware control traffic or application data) is routed (in the network) to the underlying corresponding node. Data generated and sent out by a Grid resource is treated by ns-2 as if that traffic originates at the node to which the resource is attached. The ultimate result is that the simulation of the Grid network links, packet queues in routers and transport protocols can be performed completely by ns-2 and does not need additional code.

This ns-2/NSGrid layering also allows for an adjustable level of detail when modeling the Grid's interconnecting network. Using the lowest possible level of detail, one can also model each Grid site's internal network (e.g. a switched Ethernet structure - see figure 3.2). On the other hand, it is possible to hide each Grid site's internal networking details (see figure 3.3) by only creating an ns-2 node for each site's gateway, and attaching all relevant Grid resources in this site to that same gateway.

3.3 NSGrid Models

3.3.1 Grid Model

It has already been explained in section 3.2 and shown in figures 3.2 and 3.3 that, in NSGrid, Grids are modeled as collections of interconnected and geographically

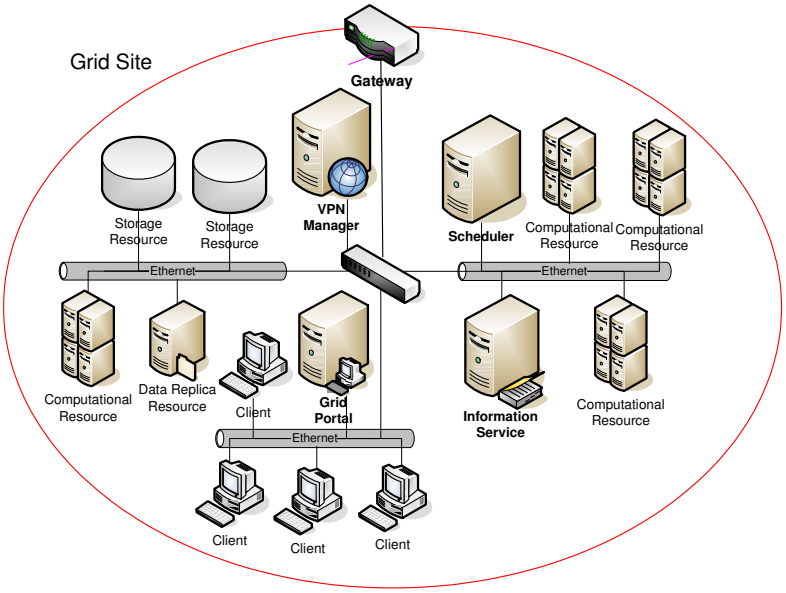


Figure 3.2: NSGrid Detailed Grid Site view with Switched Ethernet Interconnection Network

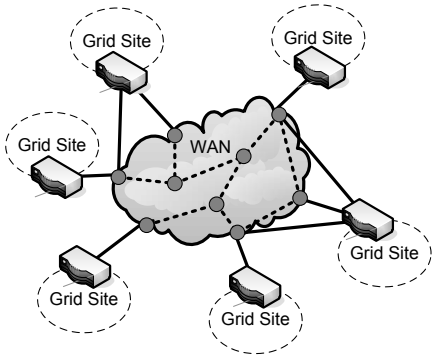


Figure 3.3: NSGrid High-Level Grid View - Grid Site internals hidden

dispersed Grid sites. Each Grid site can contain multiple resources of different types, as well as the necessary Grid middleware components to manage them.

The main types of resources are computational resources, storage resources, data repositories but also the network connecting the other resources. Resource managers (able to broker resource allocations with guaranteed QoS levels) are an important example of Grid middleware components present in NSGrid, as are job schedulers and information services. Job schedulers obtain a (partial) overview of the available resources' state, and allocate suitable resources to incoming jobs by

contacting the responsible resource managers. Jobs are submitted by clients to a scheduler using a Grid portal.

While the components representing computational and storage resources in NSGrid also implement the required resource management functionality, a dedicated connection manager is present to perform the analogous functions for a collection of network resources (as, for instance, a network path with bandwidth and delay guarantees will usually encompass multiple successive links and intermediate routers that must be allocated in advance).

3.3.2 Network Model

It has been mentioned before that the network connecting the various Grid sites and resources consists of a collection of ns-2 network elements (nodes and links). In order for these network to be treated as first-class resources by a scheduler, properties, a connection manager was added to NSGrid (see figure 3.4). Such a connection manager is responsible for a network segment, and is able to provide end-to-end bandwidth pipes to jobs when requested by a scheduler. While this bandwidth management does not translate to an existing concept in the ns-2 layer, new bandwidth allocations will fail (in the NSGrid layer) when a link on the proposed end-to-end path does not offer enough residual capacity. This way, the NSGrid layer view on the network's status is correct if the connection manager is informed of the actual capacity on each ns-2 link it manages.

The NSGrid connection manager also supports capacitated VPNs (see figure 3.5). In this mode of operation, end-to-end bandwidth pipes between Grid site pairs are set up. If jobs belong to different service classes (characterized by typical job requirements such as processing requirements, computation to communication ratios and priority), these bandwidth pipes can be reserved for a specific service class. Jobs can then only allocate bandwidth within the pipes destined for the service class they belong to. Network link capacity not allocated to a specific service type can be used by jobs from other service types in a first-come, first-served (FCFS) fashion. This mechanism is similar to the reservation of computational and/or storage resource capacity for certain service classes.

3.3.3 Computational Resource Model

The main type of computational resource supported by NSGrid is that of a time-shared processing entity, possibly featuring multiple processors. By this time-shared nature, we mean that each job can be assigned a slice (as stated in section 3.3.1, the management functionality necessary for the correct allocation of these slices has been incorporated in our NSGrid implementation of the time-shared computational resource) of the computational resource's processing power; such a slice does not change in size over its allocation lifetime.

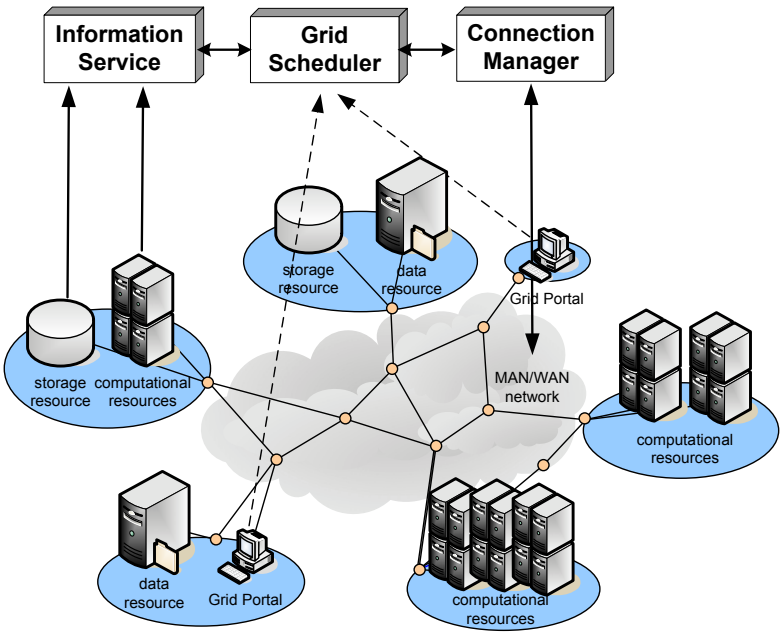


Figure 3.4: NSGrid Network Resource Management: Connection Manager Role

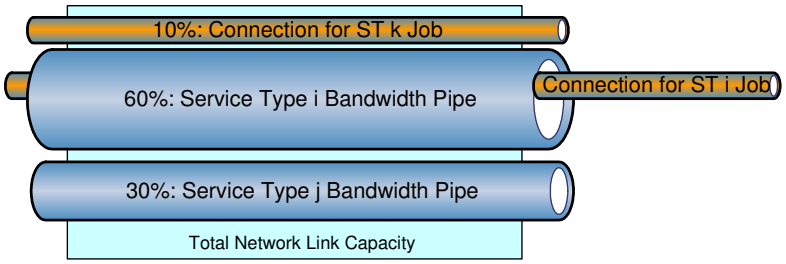


Figure 3.5: NSGrid Connection Manager: Capacitated VPN mode of operation for Service Types i and j

The main properties attributed to a NSGrid computational resource are its number of processors and their processing power (relative to the processing power of a reference processor), the maximal slice of processing power allocatable to a single job, the available memory and temporary disk space, the software environment installed on the resource and an optional cost associated with the use of this resource.

This approach allows to model both multiprocessors and clusters in NSGrid, although the latter case requires that the different cluster nodes are connected through a high-performance network (i.e. chances that this interconnecting network becomes an operational bottleneck are negligible) for the model to be ac-

curate. If this assumption is invalidated and the cluster intra-network cannot be assumed to be sufficiently performant, it is still possible to instantiate a single computational resource in NSGrid per cluster node and to explicitly model the interconnecting network.

To avoid inconsistent data to be retrieved by a Grid scheduler, a situation which can occur in scenarios where multiple schedulers operate simultaneously using a set of shared resources, resource reservations and allocations requested by a scheduler are always treated as atomic "test-and-allocate" operations: a scheduler's request to check a resource's availability either fails or results in an allocation.

Once resource allocations for a job have been made, the NSGrid computational resource's logic is responsible for the generation of suitable events to retrieve input data and store output data.

3.3.4 Storage Resource Model

In NSGrid, storage resources provide read/write disk space to jobs generating large amounts of output data. Data can remain stored until it is retrieved by the user, or until it is read by a successive job. The main model parameters describing a storage resource in NSGrid are its available storage space (possibly as a function of the class of jobs for which an allocation is requested) and the cost associated with allocating storage space.

As in the case of computational resources, storage allocations requested by a scheduler are treated as atomic "test-and-allocate" operations. In an operational Grid, storage allocations on a dedicated storage device are typically referred to using logical file names, which act like human readable file set descriptors. These files can then be retrieved using specialized bulk transfer protocols such as GridFTP [7] or remote file access schemes such as RFIO, developed for CERN's Advanced Storage Manager (CASTOR [8]).

The NSGrid storage resource does not offer computational capabilities; nevertheless, nothing prevents such a storage resource from being attached to the same ns-2 node as another Grid resource (e.g. a computational resource).

3.3.5 Data Replica Resource Model

Data replica resources provide read-only input data sets to jobs. As the name suggests, these data sets may be replicated over several such resources. Therefore, the major property of these resources in NSGrid is the set of data sets currently replicated at that resource. NSGrid data replica resources keep track of data access requests; optionally, this access log can be used by a replica manager to delete a data set from a replica or to create a new copy of a popular data set.

As with storage resources, nothing prevents a data replica resource from being co-located on the same ns-2 node as another Grid resource.

3.3.6 Resource Dynamics

Due to the large number of resources comprising a Grid, it is unrealistic to assume that each resource will be available at all times. Indeed, since resources are not managed in a centralized fashion, they may leave and join the Grid at unpredictable moments in time. The NSGrid Grid resource models (computational, storage, data replica, network) implement features that allow them to mimic periodic unavailability (due to leaving the Grid as well as due to resource failure). Currently, the NSGrid dynamic resource model is implemented using two stochastic parameters - MTTF (Mean Time To Fail) and MTTR (Mean Time To Repair) - and a calendar function used to model scheduled resource maintenance and down periods.

In addition, NSGrid is able to model job checkpointing, a mechanism allowing to resume killed jobs (e.g. due to failing resources which were allocated to that job) on a possibly different set of resources, where job state is restored from the last saved snapshot of the job's state (the checkpoint).

3.3.7 Middleware

The NSGrid simulation environment not only implements Grid resource models, but also provides a set of Grid middleware (software) component models. The NSGrid middleware components mainly implement resource management, monitoring and allocation functions. For computational and storage resources, allocation is performed by these resources themselves, while for network resources, allocations are made through the connection manager. Resource status information is gathered in a distributed set of information service components.

When attempting to schedule a job, each scheduler first retrieves status information for a suitable set of resources. Because multiple information services can be operational and return information on the same resources, each scheduler filters down the retrieved info and keeps the most recent/most plausible resource state information.

The NSGrid scheduler component is highly configurable. It can operate both in immediate mode (scheduling decisions for each job are made as soon as that job arrives) and in batch mode (scheduling decisions are made at regular time intervals for all jobs yet unscheduled). Several resource selection algorithms have been implemented in the NSGrid scheduler component. Most of these are list-based algorithms, meaning that the jobs in the ordered set of currently unscheduled jobs are scheduled one at a time until each job in this set has been processed. Job ordering can be based on the jobs' priority, service class and resource requirements; absence of ordering before starting a new scheduling operation means that incoming jobs are scheduled using a first-come, first-served (FCFS) approach. The resource selection algorithms themselves can mainly be classified based on their ability to treat network elements as first class resources (as opposed to allowing for uncer-

tainty regarding available network bandwidth between resources), their treatment of jobs belonging to different service classes and their resource locality preference.

Users submitting Grid jobs are modeled in NSGrid using client components. Clients are configured to submit jobs drawing parameters (see section 3.3.8 for a list of relevant parameters) from a predetermined range using stochastic variables. Each Grid site's portal component receives jobs from all local users and conveys them to the appropriate scheduling component.

Communication between NSGrid middleware components (i.e. job submission, resource state retrieval, resource allocation requests etc.) uses an RPC model, in which messages are encapsulated in XML structures. This XML-based approach allows to model OGSA [9] compliant Grid services (defined using the Web Services Resource Framework) in NSGrid. In addition to Grid middleware traffic, resource configuration and job description also follow an XML structured approach in NSGrid.

A detailed analysis of the NSGrid middleware component models, the XML RPC messages used and message sequence diagrams depicting the components' communication can be found in Bruno Volckaert's PhD thesis [10].

3.3.8 Application Model

In this work, we reserve the term *job* to identify atomic units of work. Thus, these jobs cannot subdivided any further, nor can they be distributed across multiple computational resources. Jobs can be constrained by precedence relations, but do not depend on (and thus, do not communicate directly with) other jobs during their execution. In this view, an *application* is then a collection of such jobs and the precedence relations existing between them. As long as no precedence relations are violated, jobs can be scheduled independently from each other.

Each job j has a length l_j, which is a measure of that job's running time on a reference computational resource, in absence of other workload on that element and in absence of bottlenecks created by other resources. Other important requirements a job can put on the computational resource selection process are the amount of available memory, the operating system and the installed software environment.

Another class of job parameters describes its input and output data sets. Jobs can process multiple input data sets and can generate multiple output data sets. These data sets are read and written in a number of equally sized *chunks*. The number of chunks into which input data set k for job j is divided is called n^i_{jk}; similarly, the number of chunks into which output data set k for job j is divided is called n^o_{jk}. We assume these input and output transfers occur in parallel with the execution of an instruction block. When the necessary input data is not available at the start of the next computational block, the job's progress comes to a temporary halt, only resuming when the data has finally arrived.

This provides a generic job model supporting both data *streaming* and *pre-staging*. If for some data set k n^i_{jk} is big (or infinite), this data set is almost continuously streamed to the job during its lifetime. On the other hand, if n^i_{jk} equals 1 for this data set, the entire data set is pre-staged to the computational resource executing the job before it is started.

The job model clearly reflects the resource allocation interdependence typical for Grid jobs: if an input data chunk is delivered late, the job's processing is suspended until the data chunk has been received. The occurrence of such processing stalls indicates that computational resource allocation (the time share allocated to that job) and Network Element allocation (bandwidth allocated to the delivery of the offending data set) are not in ideal correspondence.

A sample visual interpretation of this job model is presented in figure 3.6. In this example, job j, executed on the reference computational resource, needs 1 input data set and produces 1 output data set; n^i_{j1} is 3 while n^o_{j1} is 2. Suspension of the job's computational progress is illustrated in case the last input data chunk is delivered late.

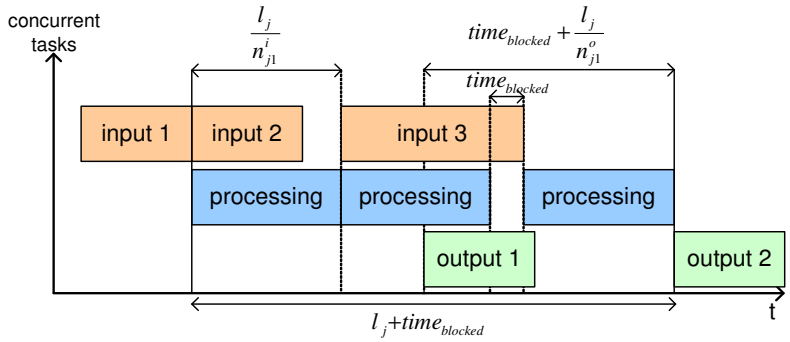


Figure 3.6: Blocking Job Model: Sequential Data Access

Note that, in absence of computational stalls, the exact run time for a job can be predicted given the capacity of the computational resource it is running on, the CPU share it receives throughout its lifetime, the bandwidth allocated between the computational resource and the involved storage resources and data repositories and the burstiness with which these data sets are read and written by the job. This case is presented in figures 3.7 and 3.8, showing both the data streaming and data staging .

Additional job properties of interest supported by NSGrid include job priority and/or service class. These properties enable schedulers to give prioritized treatment to certain classes of jobs, and to reserve resources for jobs belonging to different service classes or launched from within different VOs.

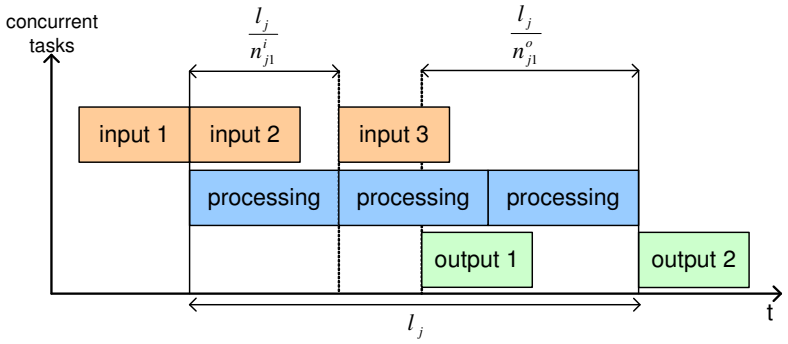


Figure 3.7: Non-Blocking Job Model: Sequential Data Access

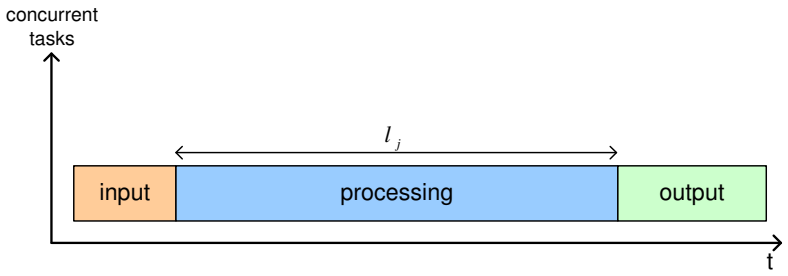


Figure 3.8: Non-Blocking Job Model: Pre-Staged Data

3.4 NSGrid: Mode of Operation

Simulating a Grid scenario using NSGrid roughly consists of the following steps, which also determine the overall structure of a typical Tcl Grid simulation driver script. First, the Grid site interconnection network is constructed using ns-2 node and link objects. Next, NSGrid Grid resources are instantiated (and their configuration is parsed from XML) and associated with the appropriate ns-2 nodes.

After resources have been instantiated, Grid middleware components are constructed and configured - at least one scheduler and one information service are required. NSGrid resources then register themselves with the information service(s).

In a last step, clients are instantiated and configured and start submitting jobs to the schedulers. During the simulation, each scheduler logs all of the resource allocations made. From these logs, it is possible to extract job response times and resource utilization.

3.5 Conclusions

In this chapter we have presented NSGrid, a Grid simulator built on top of the widely-used ns-2 network simulator. The need for a Grid simulation tool as well as the motivation to use ns-2 as a starting point have been covered extensively. We have detailed NSGrid's layered architecture as well as the different time-shared Grid resource (computational, storage, data replica and network resources) simulation models supported.

An overview of the different Grid job models supported by NSGrid was given, as well as their relationship to the different resource allocations made by a scheduler.

These schedulers are only one of the important Grid middleware components that have a corresponding NSGrid implementation: schedulers, connection managers, service managers, service monitors, information services and replication managers all have NSGrid implementations.

We have shown how NSGrid interacts with the underlying ns-2 simulator, and in a final section we detailed how a typical Grid simulation scenario is constructed and executed using NSGrid.

References

[1] P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, and P. Demeester. *Evaluation of grid scheduling strategies through NSGrid: a network-aware grid simulator*. published in Neural, Parallel & Scientific Computations, Special Issue on Grid Computing, 12:353–378, 2004.

[2] *The Network Simulator - NS2*. `http://www.isi.edu/nsnam/ns`.

[3] J. Liu, D.M. Nicol, B.J. Premore, and A.L.Poplawski. *Performance Prediction of a Parallel Simulator*. In Proc. of the Parallel and Distributed Simulation Conference (PADS'99), pages 156–164, 1999.

[4] Jason Liu, L. Felipe Perrone, David M. Nicol, Michael Liljenstam, Chip Elliott, and David Pearson. *Simulation Modeling of Large-Scale Ad-hoc Sensor Networks*. In European Simulation Interoperability Workshop, 2001.

[5] A. Varga. *OMNeT++*. IEEE Network Interactive (online), 16(4), 2002.

[6] Andras Varga. *The OMNeT++ Discrete Event Simulation System*. In Proceedings of the European Simulation Multiconference (ESM'2001), 2001.

[7] W. Allcock. *GridFTP Protocol Specification*. Global Grid Forum Recommendation GFD.20, 2003.

[8] O. Bärring, B. Couturier, J.D. Durand, E. Knezo, and S. Ponce. *Storage Resource Sharing with CASTOR*. In Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST2004), pages 345–360, 2004.

[9] I. Foster and al. *The Open Grid Services Architecture, Version 1.0*. draft-ggf-OGSA-spec-019 `http://forge.gridforum.org/projects/ogsa-wg`.

[10] B. Volckaert. *Architecturen en algoritmen voor netwerk- en dienstbewust Grid-resourcebeheer*. PhD thesis, Universiteit Gent - Faculteit Ingenieurswetenschappen, 2006.

4

Simulating Grid Scheduling Algorithms using NSGrid

4.1 Introduction

Chapter 3 has introduced our network aware Grid simulation environment called NSGrid. In this chapter, we will demonstrate the use of NSGrid in two particular cases. In the first case, we show how network aware scheduling algorithms offer an improvement over their non-network aware counterparts by means of simulating these algorithms on a Grid and comparing job response times in the resulting schedules. In the second case, we simulate the scheduling of workload on different subgrids of a partitioned Grid and use the results to evaluate the effectiveness of various Grid partitioning strategies.

4.2 NSGrid Application: Network Aware Scheduling

In sections 3.3.2, 3.3.7 and 3.3.8, we have mentioned the various network resource models supported by NSGrid, the different ways these network resources are treated by the algorithms implemented in the scheduler components and the data transfer modes supported by the NSGrid jobs.

In this section, we show how NSGrid has been used to establish the benefits of treating network elements as first-class resources. We consider the two data

transfer modes (pre-staged data and streamed data) described in section 3.3.8 as well as the different network resource allocation schemes detailed in section 3.3.2.

In particular, we focus on the average job response time obtained by scheduling a set of jobs on a fixed Grid. We compare the results obtained for different scheduling algorithms, and evaluate the set of studied scenarios for varying network link capacities.

A more thorough description of the scheduling algorithms used in these simulations, as well as comparisons using metrics differing from the average job response time and results concerning the use of capacitated VPNs can be found in [1, 2] and appendix B.

4.2.1 Grid Interconnection Topology

The results presented here in section 4.2 are obtained using a fixed Grid topology for all simulations. First, a Wide-Area Network (WAN) topology (each core router has an average out-degree of 3) has been instantiated using the *GridG* tool [3, 4]. Along the edges of this WAN topology, 12 Grid sites are instantiated. Each Grid site contains its own computational and storage resources as well as a data replica resource, connected in a LAN. This LAN is modeled as a 1Gbps Ethernet network.

Furthermore, we have homogenized the capacities of each WAN link, which we then treated as a parameter in our simulations. Each site has its own information service (storing resource properties and status) and local Grid portal through which users can submit jobs. In our setup, the Gigabit intra-site LAN provides enough capacity to ensure that no connectivity problems or bottlenecks can arise in this part of the network.

4.2.2 Grid Resource Dimensions

4.2.2.1 Computational Resources

A single computational resource has been assigned to each Grid site. To reflect the use of different tiers in existing operational Grids [5], not all computational resources are equivalent: the least powerful resource has two processors (which operate at the reference speed). A second class of computational resources has four processors, and each processor operates at twice the reference speed. The third - and last - computational resource type contains 6 processors, each of which operates at three times the reference speed.

Conversely, the least powerful type of computational resource is three times as common as the most powerful one, and twice as common as the middle one. It is assumed that all processors can be time-shared between different jobs. As jobs cannot be subdivided and distributed over multiple processors, the maximal

amount of computing power that can be allocated to a single job is determined by the capacity of the processor assigned to it.

4.2.2.2 Storage Resources

For the simulations performed, we have focused on determining the influence of the use of network resource status on the schedule calculation; therefore, we have assumed that storage resources offer unlimited disk space that can be read and written at sufficiently high speed (i.e. higher than the needed data transfer bandwidths). Each site has at its disposal exactly one such storage resource.

4.2.2.3 Data Replica Resources

Each site's data replica resource contains 6 out of 12 possible data sets. These data sets are distributed in such a way that 50% of the jobs submitted to a site can have local access to its needed data set. For each job, the data set it needs is drawn randomly from all data sets available to that job's service class using a uniform distribution.

4.2.3 Grid Jobs

We have used two different classes of job in our simulations; one is more data-intensive (i.e. higher data sizes involved), while the other is more CPU-intensive. At each Grid site, two clients have been instantiated, one for each job type. Each client submits mutually independent jobs to its Grid portal. All jobs need a single data replica resource (they process a single input data set) and a single storage resource. While the average interarrival time for jobs in each class follows the same distribution, it is assumed that the maximal data sizes and maximal reference run times differ up to a factor of 100 and 3, respectively. Obviously, the data-intensive jobs process the larger data set and the CPU-intensive jobs have the longer reference run time. Both job types make up 50% of the total job load submitted to the simulated Grid.

The simulations were performed using a batch scheduling approach, with a fixed time interval between consecutive scheduling rounds. The size of this interval was correlated to the jobs' average interarrival time in such a way that the scheduler on average finds 10 jobs in the scheduling queue at the start of each scheduling round. During the simulations, running jobs could not be pre-empted, checkpointed or moved onto a different set of resources.

4.2.4 Scheduling Algorithms

4.2.4.1 Non-Network Aware

When using a non-network aware scheduling algorithm, the Grid scheduler computes Grid job schedules based on the status of the computational, storage and data replica resources (as provided by the information services) only. Algorithms that use this kind of approach will not take into account information concerning the status of resource interconnections. The decision of which resources to use for a job will be based on the information acquired from the different information services (i.e. job execution speed and end time will be calculated based on the status of the available computational, storage and data replica resources).

It is precisely because non-network aware algorithms can only assume that residual bandwidth on network links is sufficient, that jobs can block on I/O operations: their computational progress is no longer determined solely by the computational resource's processor share that has been allocated to it (which, together with the job's length and the computational resource's relative speed determines its earliest end time if all input and output transfers complete on time, that is, before the start of the appropriate instruction block), but also by the limited bandwidth available to its input and output streams. Note that the fact that network information is discarded during the scheduling implies that no connection reservations (providing guaranteed available bandwidths) with a connection manager are made - these would allow to accurately predict the job's running time. We have used a non-network aware scheduling algorithm as a naive heuristic to compare a set of improved (network aware) algorithms to in our simulations. In our comparisons, we refer to this non-network aware algorithm as *NoNetwork*.

4.2.4.2 Network Aware

Network aware scheduling algorithms will not only contact the information services to query computational, storage and data replica resource status and availability (for those resources adhering to the job's requirements), but will also query a connection manager for information about the status of the network links interconnecting these resources. The connection manager will inform the grid scheduler about connections that can be set up between computational/data replica resource pairs (necessary for job input retrieval) and computational/storage resource pairs (needed for job output storing). Based on the answers from the information services and connection manager, the network aware scheduling algorithm is able to calculate job execution speed and end time more accurately, taking into account the speed at which input/output can be delivered to each available computational resource. For jobs with 1 input stream and 1 output stream, the best computational/storage/data replica resource triplet is the one that minimizes the expected completion time of the job. The network aware scheduling algorithms schedule ar-

riving jobs using a greedy strategy, minimizing the completion time for individual jobs as they are scheduled. This completion time is determined by the available processing power to that job on the computational resource (and its relative speed), the job's length, the job's total input and output data size and the network bandwidth that can be allocated between the involved data replica, storage and computational resources. Because network resource availability and resource allocation interdependence is taken into account, the use of a network aware scheduling algorithm allows the scheduler to calculate an accurate figure of each job's response time and to make matched allocations with each of the resources involved.

4.2.4.3 Network Aware Scheduling: Resource Locality Preference

The algorithm mentioned in the previous section only uses individual job response times as an optimization criterion. An extended algorithm has also been implemented, in an attempt to incorporate a notion of resource cost into the scheduling process.

From an economic point of view, it makes sense to only use remote (computational) resources for a job, if no computational resource close or local to the job's submission site is available. A network aware scheduling algorithm, preferring to select local resources if possible, has been implemented in NSGrid and is called *PreferLocal*. The network aware algorithm which only optimizes each job's response time, regardless of the location of the selected resources in doing so, is hereafter simply referred to as *Network*.

4.2.5 Performance Metric: Response Time

We define the *response time* of a job as the difference between its end time and the time it is submitted to the scheduler. This response time mainly consists of the time spent in the scheduler's queue and the time needed for the actual execution of the job. The execution time includes the time spent transferring input and output data from and to the computational resource allocated to the job. In case network resources are treated on the same level as computational resources, advance bandwidth allocations can be made (e.g. by the network aware scheduling algorithm), which allow for the deterministic calculation of the job's exact response time.

4.2.6 Comparison for streamed data transfer

In figure 4.1 we present this average job response time for the three algorithms we discussed earlier. In this particular simulation, simultaneous execution and data transfer were allowed; data connections were set up on a FCFS basis without upfront per service class VPN dimensioning. Clearly, for low bandwidths, not taking into account network status (when computing the schedule) incurs a

severe penalty; when bandwidth grows, the importance of this network information decreases (when the job load is constant) as the network no longer creates a bottleneck. In fact, for high bandwidths, it is possible for the non-network aware algorithm to perform slightly better; this is due to the conservative nature of the resource allocation policy used by our network aware algorithms. For instance, they assume that the maximum data transfer rate is only 95% of the available bandwidth (i.e. 5% protocol overhead) and adjust their allocations and decisions accordingly.

In our simulations, no improvement is obtained from preferring local resources. Intuitively, we expected this latter strategy to create better schedules for data-intensive jobs (as intra-site network links have high capacities). However, this possible improvement is neutralized by the asymmetry of the computational resources: jobs submitted at a site containing a slower computational resource, are less likely to be executed on a faster one (which is of course the case if the best resource collection is selected for a job) in such a scenario.

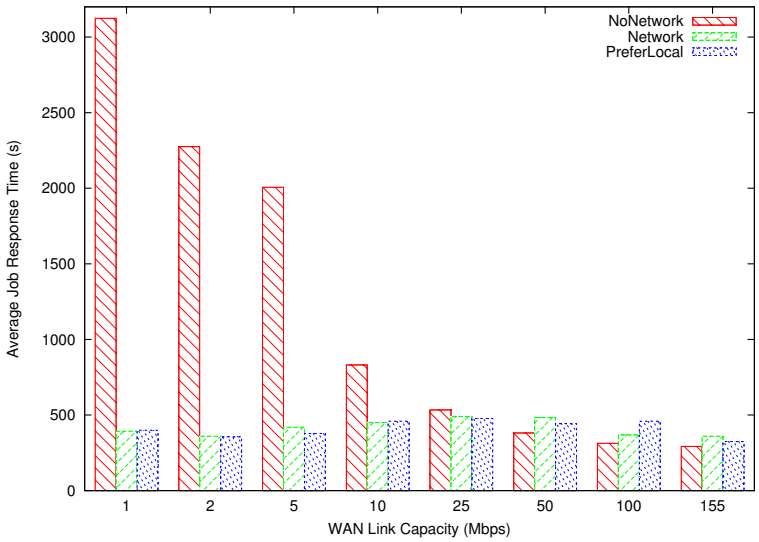


Figure 4.1: Response Time Comparison: Streamed Data

4.2.7 Comparison for pre-staged data

In figure 4.2, we have replotted the average job response time for the same job load; in this case however, jobs were not able to start executing while still downloading data (i.e. pre-staging of the entire input to the execution site was required). As the execution/transfer parallelism is lost, average response times for network aware algorithms increase. However, this loss of parallelism does not influence the relative behavior of the different algorithms (network aware or not) discussed

before. In a pre-staged scenario, network and computational resource allocations are independent. Therefore, bandwidth allocations don't need to match computational resource allocations - in contrast to a data streaming scenario, where these allocations must be matched to avoid blocking due to resource bottlenecks. In turn, this allows for a speedy data transfer prior to the start of the job processing and avoids the need for interdependent computational resource and network allocations of fixed size.

As a result, the network unaware algorithm produces better response times when pre-staging data when compared to the same algorithm scheduling the same job load using a streaming data model.

In addition, for sufficiently high bandwidths, the non-network aware scheduling algorithm performs slightly better than the network aware algorithm. This effect can be attributed to our assumption on bandwidth allocation and protocol overheads (see section 4.2.6), and it can be concluded that these assumptions are rather conservative.

The non-network aware algorithm performs a lot better for low bandwidths when using pre-staged data (when compared to the use of streaming data), which seems counterintuitive at first. We believe, however, that this is mainly caused by two NSGrid implementation choices. First of all, data transfer start times for all data blocks are calculated as the latest possible start times that allow the data to arrive in time. If these start times are based solely on the size of the computational resource slice allocated to a job (without matching this slice with the available network bandwidth), the time needed to transfer data will be underestimated and the possible execution/data transfer parallelism due to the use of a streaming data model will be lost to a great extent due to blocking. Furthermore, when using the streaming data model, the NSGrid scheduler co-allocates network and computational resources. This means that the computational resource is also allocated while the job is blocking due to limited network bandwidth, a phenomenon that does not occur when using the pre-staged data model. In turn, this means that scheduling (in a non-network aware fashion) a data-intensive job to use a remote data repository using the streaming data model will significantly reduce the Grid's available computing power when compared to the same job being scheduled in the pre-staged data setting.

4.3 NSGrid Application: VPG Resource Partitioning

In section 4.2 we demonstrated, using a sample Grid simulation scenario, the effectiveness of network resource allocation strategies (and, in particular, the added value of manageable network resources in a Grid setup) by means of NSGrid. In

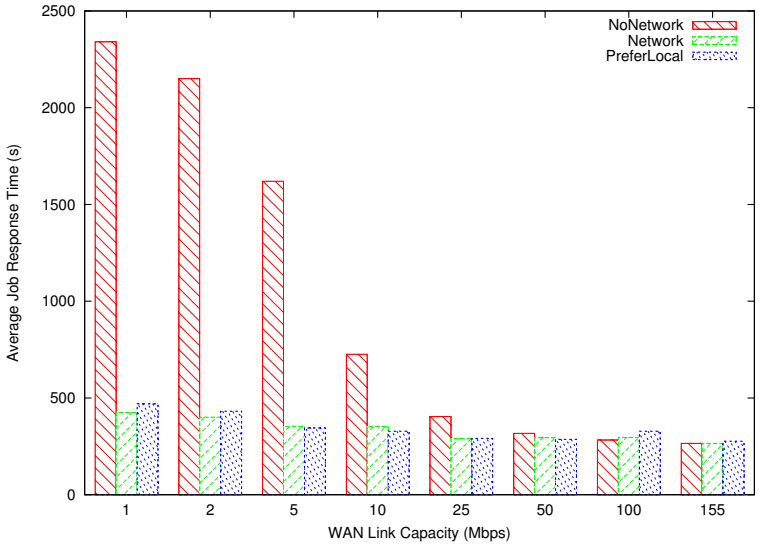


Figure 4.2: Response Time Comparison: Pre-Staged Data

addition, we showed results regarding the influence of pre-allocated network bandwidth to different job service classes.

In this section, we extend this approach to the entire range of Grid resources: we pursue and investigate the partitioning of a set of Grid resources (of various types) to certain job service classes, with the goal to reserve the use of the resources inside a particular element of such a partition exclusively to jobs of the intended service class. The net result is a collection of so-called *Virtual Private Grids*. We study several algorithmic approaches to calculating suitable resource partitioning schemes and compare their complexity. The partitioning also affects scheduling decisions, as schedulers need only to investigate a smaller search space when resources have been partitioned. Therefore, we investigate scheduling performance of several algorithms introduced in section 4.2.4 on a partitioned Grid.

The VPG partitioning algorithms described here, as well as the service management architecture (responsible for the partitioning) implemented in NSGrid have been detailed in [6] as well as in Bruno Volckaert's PhD thesis (see section 1.3), and we refer the reader to that paper for extended simulation results and detailed comparisons of the algorithms mentioned in this section.

4.3.1 VPG Partitioning

As more and more application types are ported to Grid environments, an evolution is noticed from purely computational and/or data Grid offerings to full-scale service Grids [7] (e.g. the EGEE Enabling Grids for E-Science in Europe project [8]).

Such a service Grid is actually a Grid infrastructure capable of supporting a multitude of *application types* with varying QoS levels and resource requirements, and the term should not be mistaken to simply mean a web services enabled Grid - although architectural standards for service Grids are provided by the Global Grid Forum's Open Grid Service Architecture (OGSA) [9] and (to a lesser extent) the Web Service Resource Framework [10], building on concepts of both Grid and Web Service communities.

In such a service Grid, user-submitted Grid jobs that exhibit similar resource requirements (such as processing requirements, I/O data requirements and priority) can be classified according to these resource requirements and tagged with a *service class*. By correctly identifying service classes (based upon monitoring), Grid resources (computational, storage, data replica and network resources) can be partitioned into subsets, each subset being allocated exclusively to jobs belonging to a single service class. The end results is that for each service class a *Virtual Private Grid* (VPG) is established. The partitioning of a Grid into these VPGs is thus a case of making well-informed advance resource reservations. The use of VPGs offers a series of benefits: once partitioned, VPGs can be managed individually and independently, improving resource management scalability, and each VPG's management policies can be specifically tuned for the service class they deal with. In addition, faster scheduling decisions are made possible because fewer resources need to be examined.

4.3.2 VPG Partitioning Support in NSGrid

In NSGrid, a distributed service management architecture has been implemented as a set of software components providing resource-to-service partitioning strategies and aiding in the deployment of Virtual Private Grids. Within each VPG, information services and schedulers can be instantiated as standard NSGrid components.

The dynamic VPG partitioning is implemented in NSGrid using two additional components: a Grid service monitoring component, responsible for acquiring and extracting service class properties, and a service management component which executes the actual VPG partitioning algorithms, using the acquired monitoring data as input. When changes are detected in the service characteristics by the monitoring component, a new VPG partitioning action may be triggered.

The resulting service management architecture thus closely mimics the OGSA *Service Level Manager* concept. Service Level Managers are, according to the OGSA specification, responsible for setting and adjusting policies, and changing the behavior of managed resources in response to observed conditions.

A Grid, partitioned into VPGs by the implemented service management architecture, looks roughly like the one shown in figure 4.3 in NSGrid.

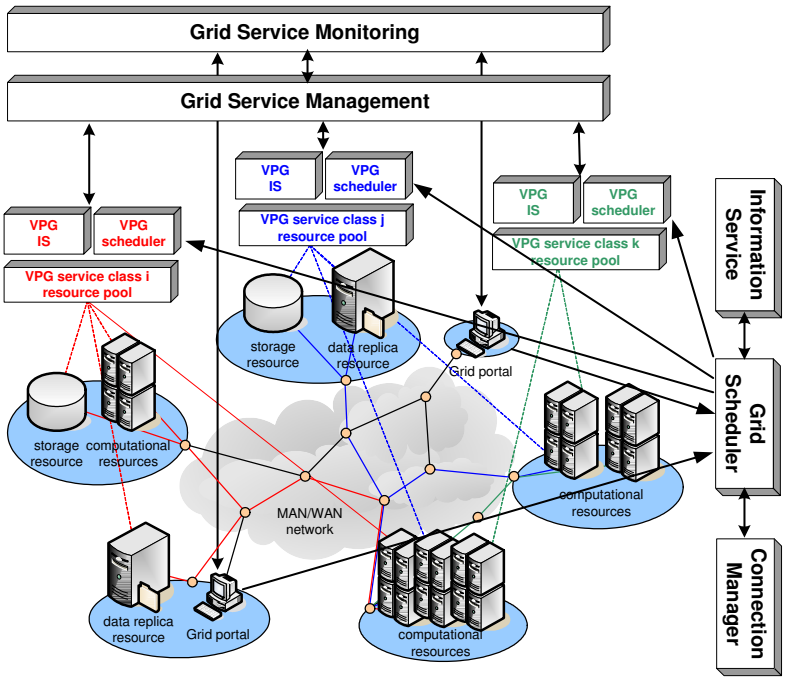


Figure 4.3: VPG Partitioned Grid

4.3.3 Partitioning Strategies

4.3.3.1 Genetic Algorithm

In order to solve the VPG partitioning problem using a genetic algorithm (GA), note that each possible resource class assignment can easily be represented as an n-tuple of service class IDs, where n equals the number of resources. These *chromosomes* can then be fed to a genetic algorithm which evaluates the fitness of each chromosome (i.e. possible service class assignment) with regard to a cost function $f(x)$.

A genetic algorithm performs selections, cross-overs and mutations, with the goal of constructing better resource-to-service assignments (that is, when compared using cost function $f(x)$). The algorithm starts using a randomly generated set of such assignments, and stops after a configurable number of iterations has been reached or when no significant improvement in resource-to-service assignment quality is detected for a number of successive iterations.

Using appropriate cost functions, a number of different assignment strategies can be explored. One strategy focusses on providing adequate resource capacity close or local to the site where the bulk of a service class's workload originates; it attempts to distribute resource capacity fairly among these local service classes.

A second strategy allows to weigh the importance of these local service classes against the importance of other service classes, resulting in the cost function depicted in algorithm 4.3.1. To promote the concept of close computational and data replica resources, cost functions include an adjustable term to penalize scenarios in which a site's computational resource is allocated to some service class, while that same site's data replica resource does not contain data sets needed by jobs of that service class.

Algorithm 4.3.1: $f_{CRpart_{global}}(x)$

$result \leftarrow \frac{ppower_{asg_0}}{2}$
$maxAlloc_{over} \leftarrow 0$
$maxAlloc_{under} \leftarrow 0$
for $i \in SC_{local} \cup SC_{foreign}$
do
- $aux \leftarrow ppower_{req_i} - ppower_{asg_i}$
- **if** $aux < 0$
  - **then**
    - **if** $-aux > maxAlloc_{over}$
      - **then** $maxAlloc_{over} \leftarrow -aux$
    - $aux \leftarrow ppower_{asg_i}$
  - **else**
    - **if** $\frac{aux}{ppower_{req_i}} > maxAlloc_{under}$
      - **then** $maxAlloc_{under} \leftarrow \frac{aux}{ppower_{req_i}}$
    - $aux \leftarrow ppower_{asg_i} - aux$
- **if** $i \in SC_{foreign}$
  - **then** $aux \leftarrow aux \times \rho_{SC_{foreign}}$
- $result \overset{+}{\leftarrow} \frac{priority_i}{(\sum_{j \in SC} priority_j)} \times aux$

$result \overset{-}{\leftarrow} maxAlloc_{over} + maxAlloc_{under}$
return $(result)$

A first set of partitioning algorithms in NSGrid mainly performs computational resource partitioning as its core functionality, but extensions have been provisioned to enable weighted network bandwidth partitioning as well.

4.3.3.2 Divisible Load Integer Linear Programming

When a Grid operates in steady state, the arriving workload can be characterized (with regard to interarrival times, processing and I/O requirements) using a limited set of parameters, representing the expected values (in the steady state) of the

observed variables, commonly represented as the workload (of which the exact definition depends on the type of resources one is observing) arriving per time unit. In NSGrid, this characterization is performed by the service monitoring architecture. The resulting parameters can then be used to populate an integer linear program (ILP) designed to calculate a feasible VPG partitioning scheme as well as the workload distribution in the resulting VPGs. The optimization goal can then be related to resource cost and utilization, under the constraint that as much arriving workload as possible must be processed by the partitioned Grid.

By treating the steady state workload as arbitrarily divisible (see chapter 2) - and thus represent it using continuous parameters - the number of integer variables in the ILP can be greatly reduced.

Using this approach, values of interest are $arrivals_s^n$ - the computational load per time unit arriving at site s and belonging to service class n, $Sets_n$ and $Size_n$ - the datasets available to service class n jobs and their respective sizes. It is assumed that each job reads exactly one input data set and that all data sets for a single service class are equally probable. The main decision variables in the ILP are $x_{c,n}$ (binary, assigning service class n exclusive access to CR c) and $\alpha_{i,n}^c$ (real-valued, amount of service class n computational load per time unit processed at CR c which arrived at site i). Auxiliary variables needed to fulfill routing constraints on the input datasets and generated output data have been dubbed $in_{n,j}^l$ (bandwidth needed on link l for transport of dataset j of service class n) and out_s^l (bandwidth needed on link l for transport of output data to storage resource s) - note that the concept of source-based routing [11] was used to formulate the routing constraints.

In the resource-to-service assignation ILP below, the following auxiliary symbols have been used: GW represents the set of Grid site gateways, L^+ depicts the set of links incident from the node specified in subscript, L^- depicts the set of links incident to the node specified in subscript and computational resource and network link capacities are written Cap_c and Cap_l, respectively. The sets of service classes, computational resources, storage resources and data replica resources have been abbreviated as SC, CR, SR and DR. For each resource r in one of these classes, we will use $Site_r$ to denote the Grid site the resource is located at.

Using these symbols, the capacity constraints to be observed for each computational resource and network link, respectively, are

$$\forall c \in CR. \sum_{i \in Sites} \sum_{n \in SC} \alpha_{i,n}^c \leq Cap_c \tag{4.1}$$

$$\forall l \in L. \sum_{n \in SC} \sum_{j \in Sets_n} in_{n,j}^l + \sum_{s \in SR} out_s^l \leq Cap_l \tag{4.2}$$

These constraints ensure that work allocated to a computational resource does not exceed that resource's processing capacity, and that total network traffic over each link does not exceed that link's capacity. Network traffic is routed according to

following constraints:

$$\forall n \in SC, j \in Sets_n. \sum_{d \in DR: j \in Sets_d} \sum_{l \in L_d^+} in_{n,j}^l = \frac{\sum_{s \in Sites} arrivals_s^n \times Size_n}{\#Sets_n} \tag{4.3}$$

$$\forall c \in CR, n \in SC, j \in Sets_n. \sum_{l \in L_c^-} in_{n,j}^l = \frac{\sum_{i \in Sites} \alpha_{i,n}^c \times Size_n}{\#Sets_n} \tag{4.4}$$

$$\forall c \in CR, s \in SR. \sum_{l \in L_c^+} out_s^l = \sum_{n \in SC} \alpha_{Site_s,n}^c \times Size_n \tag{4.5}$$

$$\forall s \in SR. \sum_{l \in L_s^-} out_s^l = \sum_{n \in SC} arrivals_{Site_s}^n \times Size_n \tag{4.6}$$

$$\forall g \in GW, n \in SC, j \in Sets_n. \sum_{l \in L_g^-} in_{n,j}^l = \sum_{l \in L_g^+} in_{n,j}^l \tag{4.7}$$

$$\forall g \in GW, s \in SR. \sum_{l \in L_g^-} out_s^l = \sum_{l \in L_g^+} out_s^l \tag{4.8}$$

The first equation in this series describes how much traffic is carried on the network links departing from the data replica resources, given that any job of a given service class has an equal probability to process any of the data sets available to that service class. That same amount of network traffic is of course to be retrieved at the computational resource side, and this is expressed in the second equation.

The next two equations present the analogous observation for output data generated by the jobs.

The last two equations state that network flow (both for input and output data) is conserved when crossing intermediate routers.

A feasible schedule is obtained by demanding that the total distributed workload equals the size of the arriving workload per time unit:

$$\forall i \in sites, n \in SC. \sum_{c \in CR} \alpha_{i,n}^c = arrivals_i^n \tag{4.9}$$

To ensure the exclusive reservation of each computational resource, we need to enforce

$$\forall c \in CR. \sum_{n \in SC} x_{c,n} \leq 1 \tag{4.10}$$

$$\forall c \in CR, n \in SC. \sum_{i \in Sites} \alpha_{i,n}^c \leq x_{c,n} \times Cap_c \tag{4.11}$$

where the last equation is used to express that only those computational resources which have been explicitly assigned to a service class may actually perform work in that service class.

The cost function to be minimized can take several forms; for instance, the total amount of data traveling over network links per unit of time (in the steady state Grid) can be described in terms of problem variables as

$$\sum_{l \in L} \left(\sum_{n \in SC, j \in Sets_n} in_{n,j}^{l} + \sum_{s \in SR} out_{s}^{l} \right) \tag{4.12}$$

Using this cost function in the ILP results in a workload schedule and service class assignation yielding minimal aggregate network load for a given arrival process. Alternatively, one can choose to minimize the maximal unused computational resource fraction, which results in a fair workload distribution across all computational resources according to their respective capacities. This approach can be modeled by adding the constraints

$$\forall c \in CR, n \in SC.cost \geq \frac{\left(x_{c,n} \times Cap_c - \sum_{i \in Sites} \alpha_{i,n}^{c}\right)}{Cap_c} \tag{4.13}$$

and minimizing the cost. Note that, whatever cost function is used, one is limited to using linear expressions in the problem variables, yielding less expressive power when compared to the genetic algorithm approach.

4.3.4 Grid Topology

The modeled Grid on which our VPG partitioning experiments have been performed (using NSGrid) closely resembles the Grid described in section 4.2.1. The WAN core network interconnecting the Grid sites is the same GridG-generated network. Again, 12 Grid sites have been instantiated, each featuring its own computational, storage and data replica resources connected through a 1Gbps Ethernet LAN.

Not all time-shared computational resources are equivalent: we have used the same three types of computational resources as listed in section 4.2.2.1, where the least powerful CR has two processors (operating at a reference speed), the second class of computational resources has four processors (operating at twice the reference speed) and the third type contains 6 processors, each of which operates at three times the reference speed. As in the network aware scheduling experiments, the relative occurrence frequency of the different computational resource types is $1 : 2 : 3$, the most powerful resource being the most rare as well.

The assumptions on each site's storage and data replica resources also remain valid for these experiments: storage resources offer unlimited disk space, but reside on bandwidth-limited nodes. Furthermore, each site's data resource contains 6 out of 12 possible data sets, distributed as to provide half of the jobs with local access to their needed data set.

Specifically for the VPG partitioning, a single global service manager was instantiated; as the service characteristics in a simulation setup are readily available (and can be fed directly to the service manager), we did not instantiate a monitoring component.

4.3.5 Performance Metric: Job Response Time

For the comparison of the different VPG partitioning strategies, we have first partitioned the Grid based on the workload described below. Then, we have scheduled this very workload on the partitioned Grid and measured the average job response time, again defined as the difference between the time the job's final output block has been sent and the time it was submitted to the scheduler. We have measured and plotted this average job response time for the different partitioning algorithms described here.

4.3.6 Job Workload

As with our experiments described in section 4.2, we have used two different service classes of equal priority, each accounting for half of the total job load. One service class represents data-intensive (i.e. higher data sizes involved) jobs, while the other represents the more CPU-intensive jobs. At each Grid site, two clients (one for each job service class) submit mutually independent jobs to its Grid portal. All of these jobs need a single data replica resource and a single storage resource.

The relevant job parameters used have been listed in table 4.1.

	CPU-Job	**Data-Job**
Input(GB)	0.01-0.02	1-2
Output(GB)	0.01-0.02	1-2
IAT(s)	30-40	30-40
Ref. run time(s)	100-200	40-60

Table 4.1: Job Workload: Key Parameters

4.3.7 Results

We have presented the measured average job response time for the different VPG partitioning strategies and the different job scheduling strategies implemented in NSGrid in figures 4.4 and 4.5. Figure 4.4 shows the response time obtained by the non-network aware scheduling algorithm, while figure 4.5 shows the corresponding results obtained by using the network aware job scheduling strategy (see section 4.2.4). The job scheduling algorithms used schedule jobs in a greedy fashion, minimizing the resulting completion time for each individual job.

Each bar in the figures points to a different VPG partitioning algorithm used in that scenario; we have shown the overall average as well as the average response times within each individual job service class. The bar titled "No SM" (no service management) points to the case where we refrained from partitioning the initial Grid topology. The genetic algorithm computational resource partitioning approach corresponds to the "GA" bar; its extended counterpart which also partitions network resources is referred to in the figures as "GA-CONN". The same Grid has also been partitioned using the divisible load integer linear programming method, using one of the cost functions 4.12 and 4.13. The average job response times for the workload scheduled on the Grid, partitioned using these approaches, are shown as "DLT CR" and "DLT Network", respectively.

From these figures, it follows that in the given Grid scenario average job response times can be improved significantly (by 40.44% when non network aware scheduling is used and by 22.6% when network aware scheduling is employed) by performing a resource-to-service partitioning prior to scheduling. The main cause for this behavior is the fact that, once the Grid has been partitioned, resources are reserved for exclusive use by a single service class. It is this service-exclusivity that forces the scheduler to not assign jobs to less-than-optimal resources (e.g. non-local access to input data, low processing power available,...), but to keep the job in the scheduling queue instead until suitable (i.e. assigned to the job's service class) resources become available.

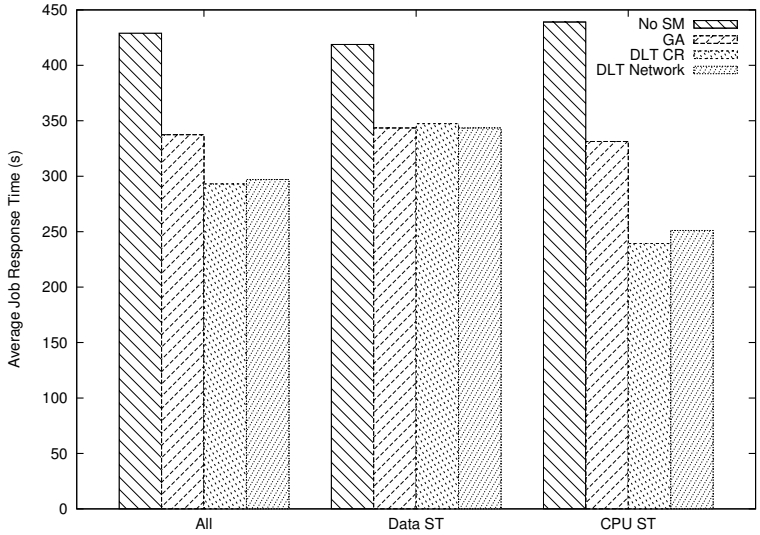


Figure 4.4: Non-Network Aware Scheduling: Job Response Times after VPG Partitioning

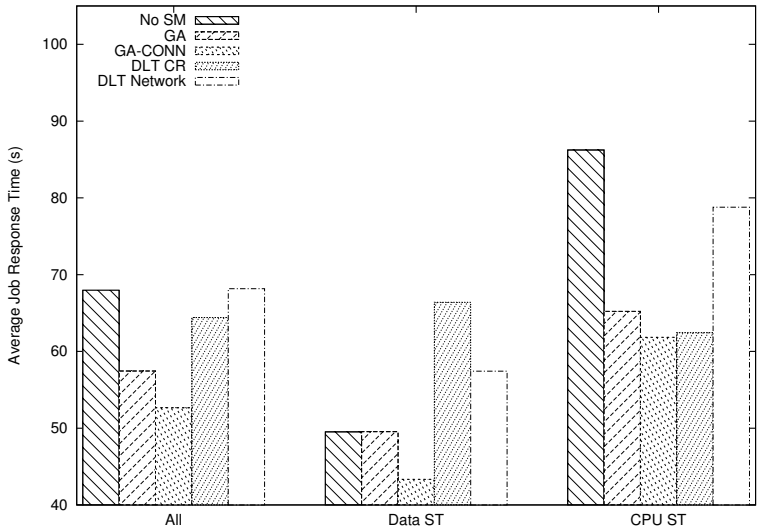


Figure 4.5: Network Aware Scheduling: Job Response Times after VPG Partitioning

4.4 Conclusions

In this chapter, we have demonstrated our NSGrid simulation tool in two cases. The first case highlighted the improvement in job response time when using a network aware Grid scheduling algorithm over a non-network aware algorithm, both when streamed data transfers and pre-staged data setups are used. The importance of using a network-aware scheduling strategy has been demonstrated in scenarios where network bandwidth cannot be assumed to be abundant. Furthermore, we introduced a scheduling strategy taking into account data location (based upon economic principles), but did not notice improvements to the job response times when using this strategy on our scenario. In the second case, we simulated workload scheduling on a partitioned Grid in which resources can be exclusively reserved for specific job service types. We showed how, for a workload scenario featuring 2 distinct job classes (CPU-intensive and data-intensive), scheduling on a suitably partitioned Grid reduces job response times. We presented several classes of Grid partitioning algorithms, one class of algorithms based on genetic algorithms and another one based on divisible load theory. We showed how there exists a trade-off between computation time and partitioning quality for these algorithm classes, and found that the genetic algorithm based approach yields better Grid partitioning schemes at the expense of increased computation time.

References

[1] P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, and P. Demeester. *Network Aspects of Grid Scheduling Algorithms*. In Proceedings of PDCS 11, pages 91–97, 2004.

[2] B. Volckaert, P. Thysebaert, M. De Leenheer, F. De Turck, B. Dhoedt, and P. Demeester. *Network Aware Scheduling in Grids*. In Proceedings of 9th European Conference on Networks & Optical Communications (NOC), pages 311–318, 2004.

[3] D. Lu and P. Dinda. *Synthesizing Realistic Computational Grids*. In Proceedings of ACM/IEEE Supercomputing 2003 (SC 2003), page 16, 2003.

[4] D. Lu and P. Dinda. *GridG: Generating Realistic Computational Grids*. ACM SIGMETRICS Performance Evaluation Review, 40(4), 2003.

[5] *The DataGrid Project*. `http://eu-datagrid.web.cern.ch/eu-datagrid/`.

[6] B. Volckaert, P. Thysebaert, M. De Leenheer, F. De Turck, B. Dhoedt, and P. Demeester. *Flexible Grid service management through resource partitioning*. Journal of Supercomputing, 2006. Accepted for Publication.

[7] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. *Grid services for distributed system integration*. IEEE Computer, 35(6):37–46, 2002.

[8] *Enabling Grids for E-Science in Europe*. `http://egee-intranet.web.cern.ch`.

[9] I. Foster and al. *The Open Grid Services Architecture, Version 1.0*. draft-ggf-OGSA-spec-019 `http://forge.gridforum.org/projects/ogsa-wg`.

[10] K. Czajkowski and al. *The WS-Resource Framework Version 1.0*. `http://www.globus.org/wsrf/specs/ws-wsrf.pdf`.

[11] Y. Kitatsuji, K. Kobayashi, Y. Kitamura, and al. *Deployment of APAN Tokyo XP and evaluation of source based routing*. Transactions of the Institute of Electronics, Information and Communication Engineers, J85-B:1164–1171, 2002.

5

Scalable Lambda Grid Dimensioning

5.1 Introduction

In chapter 1, we have introduced the concept of Grids and we have made the case for the use of optical technologies in the network interconnecting the various Grid resources.

Prior to Grid deployment, Grid resource locations and capacities must be decided upon. This lambda Grid [1–4] dimensioning problem builds on both the workload scheduling problem and the network routing and dimensioning problems introduced in chapter 2.

The lambda Grid dimensioning problem differs from these two related problem classes, however. First of all, it differs from a traditional (project) scheduling problem [5–8] because multiple resources of different types must be co-allocated simultaneously to a single job *and* because these allocations are not independent (e.g. computational progress can be limited by lack of network bandwidth in case two remote resources are allocated to a single job).

Secondly, as we are specifically dealing with lambda Grids (and thus, optical transport networks), additional network constraints reflecting wavelength continuity and granularity arise when compared to the more simple continuous-bandwidth network scenarios used in chapter 4. But in addition, the lambda Grid dimensioning problem differs from an OCS network dimensioning problem with static demand matrix [9–13] because in a Grid environment, network traffic is generated by Grid jobs. Thus, the amount of network traffic generated in the Grid and its desti-

nation fundamentally depend on the way resources are allocated to jobs. Network traffic then comes into existence because two geographically dispersed resources have been co-allocated to a single job.

The exact identity of the resources allocated to each job is decided upon by a scheduler. It follows that the lambda Grid dimensioning problem is inextricably entangled with the workload scheduling problem in the envisioned Grid.

In this chapter, we start by detailing the Grid and workload models used throughout this chapter and we identify the operational scenarios that must be supported by the resulting dimensioned Grid. We explicitly focus our dimensioning efforts on two types of resources: computational resources and network resources. As we primarily deal with optical circuit switched networks, the latter resources are actually entities such as fibers, wavelengths and cross-connects.

Our dimensioning approach consists of two steps; in the first step, we derive appropriate dimensions for the computational resources based on the expected workload characteristics. In the second step, the workload for which the Grid is to be dimensioned is scheduled on these computational resources, and sufficient network resources are instantiated to support this operation.

This second step, the solving of a combined workload scheduling and network dimensioning problem, is performed by modeling this problem as an integer linear program. To reduce this program's complexity, two techniques described in chapter 2 are used. The first technique corresponds to the treatment of the Grid's workload as an arbitrarily divisible workload [14–16]. The second technique reduces the complexity of the network routing problem [13] in case the exact nature of possible wavelength conversions in the network's cross-connects does not need to be determined. The objective of this linear program leading to the Grid's network resources' dimensions is to minimize the cost associated with these network resources, in particular the number of activated fibers and wavelengths in the envisioned operational scenarios.

Next, we present some heuristics which can further improve the time needed to solve the dimensioning problem (i.e. the linear program). We compare all of our solution methods on a set of Grid interconnection networks for varying parameters including Grid scheduling policy, wavelength granularity and fiber/wavelength activation cost models.

While we start with a set of operational scenarios in which a single Grid site is overloaded with jobs, we extend our approach to scenarios with multiple overloaded sites and scenarios featuring single resource failures afterwards.

Research results presented in this chapter have been published for a great deal in [17, 18].

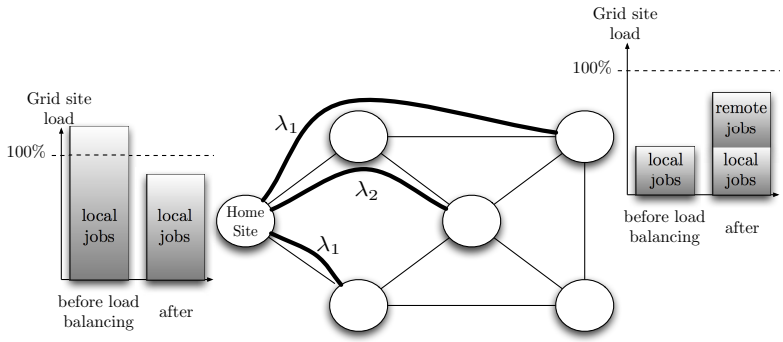


Figure 5.1: Example load balancing scenario

5.2 Grid Models and Operational Scenario

The Grid resource and application models supported by our simulation environment have been detailed in chapter 3. In this section, we introduce some notations and mathematical symbols that will allow us to represent these resources and applications in a linear program describing a lambda Grid dimensioning problem for the scenarios presented in section 5.2.3.

5.2.1 Resources

We treat a Grid as a collection of different sites $\mathcal{R}$, connected through a transport network. The core network (which is to be dimensioned) is an optical circuit switched transport network. It consists of core and access optical cross connects (OXC) connected through directed links from the set $\mathcal{E}$. Each link $e \in \mathcal{E}$ contains optical fibers; each fiber can carry a (technology-dependent) number of wavelengths W, and each wavelength supports a (also technology-dependent) data rate B. All cross connects have unlimited wavelength conversion capabilities (see section 2.4 for details on OXC switching limitations).

Each Grid site $r \in \mathcal{R}$ connects to an access router of the optical network and offers two time-shared resources - a computational resource and a data storage resource. The computational resource can process locally submitted as well as "foreign" jobs, and has a residual computational capacity of P_r. It will only send locally generated jobs to a remote site if it cannot process or store that job locally. The data storage resource holds input and output data for jobs; it is assumed they provide sufficient storage space for the jobs submitted at the resource's site.

5.2.2 Jobs

At each site, users can submit jobs from a job pool $\mathcal{J}$. The *home site* of a job $j \in \mathcal{J}$ is the site where it has been submitted. Jobs are indivisible work packets, characterized by their length l_j (i.e. processing time on a reference processor), the size of the input data d_j^I they process and the size of the output data d_j^O they generate. It is assumed that all jobs read their input data from their home site and that they submit any output data to their home site as well, that is, only remotely processed jobs produce network traffic (between their processing site and their home site). Furthermore, jobs are assumed to process data at a constant rate throughout their lifetime; this way, remotely executed jobs can be treated as Constant Bit Rate (CBR) sources from a network point of view.

5.2.3 Excess Load Scenarios

In our approach, computational resources are first dimensioned to be able to deal with a specified steady state load. The exact computational resource dimensions are derived as described in section 5.3.6. Next, we assume that a single computational resource suffers from excessive (locally generated) load and that it needs to invoke remote computational resources to process that load.

We consider the set (parameterized by some integer k) of load-balancing scheduling strategies where the excess load is evenly distributed across k remote computational resources, a scenario not unlikely given that these remote resources may also be processing or storing local load. Again, we assume the Grid to converge into a steady state mode of operation (e.g. periodic with period T). For a given excess load instance per time-period, we can decide which jobs are to be processed where (under the constraint of fair distribution across all remote resources), which determines the amounts of input and output data transferred per period between Grid sites.

Once traffic demands between each pair of Grid sites have been determined, solving the optical network dimensioning problem (for this single overloaded Grid site scenario) means deciding how lightpaths should be set up and routed in order to accommodate these demands with minimal cost. Here, only activation costs (fiber and wavelength) are taken into account.

The final network dimensions (i.e., number of installed fibers on each link and number of wavelengths activated on each fiber) are determined by the global optimum over all single-site overload problems.

5.3 Lambda Grid Dimensioning Algorithms

In the following, we present several approaches to the lambda Grid dimensioning problem. More specifically, section 5.3.1 introduces an ILP formulation for the

exact workload; parallel and incremental reformulations (reducing its complexity) of the problem are presented in the following section. The complexity of the exact workload ILP model is reduced for large job counts in section 5.3.4. Finally, the DLT-based approach is given in section 5.3.5. Each approach results in a linear program, which is solved as described in section 5.4.1. The common objective of all linear programs is the minimization of the network cost, which is defined as the weighted sum of the number of activated fibers and the number of used wavelength channels.

5.3.1 Exact Workload ILP

5.3.1.1 Single Scenario Formulation

The following ILP formulation allows us to optimally dimension the optical network for a single overloaded resource S. Suppose the excess job load of this resource is given explicitly by a set of jobs $\mathcal{J}^S$, and that a time horizon T is envisioned for the execution of these jobs (for instance, set $\mathcal{J}^S$ makes up one period of a periodically recurring job load). We introduce the following integer variables (see figure 5.2 for an illustration of their physical interpretation):

- f_e^S = number of fibers on directed edge e,
- c_{er}^S = number of wavelengths originating from resource r carried by edge e, with $0 \leq c_{er}^S \leq \left\lceil \sum_{j \in \mathcal{J}^S} \frac{d_j^I}{BT} \right\rceil$ for $r = S$ and $0 \leq c_{er}^S \leq \left\lceil \sum_{j \in \mathcal{J}^S} \frac{d_j^O}{BT} \right\rceil$ for $r \in \mathcal{R} \backslash \{S\}$,
- $y_{jr}^S = 1$ iff job j is executed on resource r, 0 otherwise,
- d_{uv}^S = demand (number of required end-to-end wavelengths) between resources u and v, with $0 \leq d_{uv}^S \leq \left\lceil \sum_{j \in \mathcal{J}^S} \frac{d_j^I}{BT} \right\rceil$ for $u = S$ and $v \in \mathcal{R} \backslash \{S\}$, $0 \leq d_{uv}^S \leq \left\lceil \sum_{j \in \mathcal{J}^S} \frac{d_j^O}{BT} \right\rceil$ for $u \in \mathcal{R} \backslash \{S\}$ and $v = S$, $d_{uv}^S = 0$ otherwise.

The first set of constraints ensures that all excess jobs are remotely executed, while protecting each resource from being overloaded:

$$\forall j \in \mathcal{J}^S . \sum_{r \in \mathcal{R}} y_{jr}^S = 1 \tag{5.1}$$

$$\forall j \in \mathcal{J}^S . y_{jS}^S = 0 \tag{5.2}$$

$$\forall r \in \mathcal{R} . \frac{\sum_{j \in \mathcal{J}^S} l_j y_{jr}^S}{T} \leq P_r \tag{5.3}$$

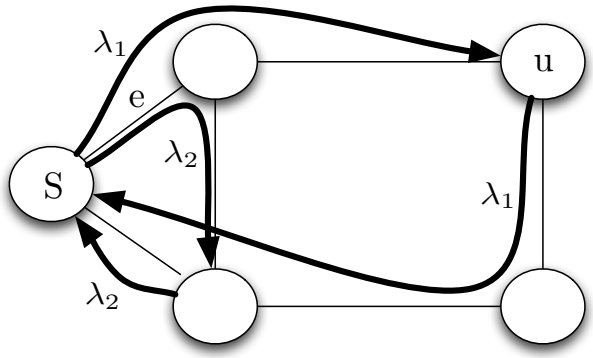


Figure 5.2: Example 5-node network with $d^S_{Su} = d^S_{uS} = 1$ and $c^S_{eS} = 2$.

Next, the demand variables are bound by the CBR traffic generated by each job:

$$\forall r \in \mathcal{R}\backslash\{S\}.d^S_{Sr} \geq \frac{\sum_{j\in\mathcal{J}^S} d^I_j y^S_{jr}}{BT} \tag{5.4}$$

$$\forall r \in \mathcal{R}\backslash\{S\}.d^S_{rS} \geq \frac{\sum_{j\in\mathcal{J}^S} d^O_j y^S_{jr}}{BT} \tag{5.5}$$

The following constraints express the network flow conservation ($\mathcal{E}^+_v$ is the set of outgoing directed links from resource v, $\mathcal{E}^-_v$ the set of incoming links):

$$\forall u \in \mathcal{R}, \forall v \in \mathcal{R}\backslash\{u\}. \sum_{e\in\mathcal{E}^+_v} c^S_{eu} + d^S_{uv} = \sum_{e\in\mathcal{E}^-_v} c^S_{eu} \tag{5.6}$$

$$\forall r \in \mathcal{R}. \sum_{u\in\mathcal{R}} d^S_{ru} = \sum_{e\in\mathcal{E}^+_r} c^S_{er} \tag{5.7}$$

Finally, connections carried on an edge force the activation of fibers on that particular edge:

$$\forall e \in \mathcal{E}. \sum_{r\in\mathcal{R}} c^S_{er} \leq W f^S_e \tag{5.8}$$

Our goal is to minimize the network cost, which is given by:

$$\sum_{e\in\mathcal{E}} (\alpha f^S_e + \beta \sum_{r\in\mathcal{R}} c^S_{er})$$

However, to model the fair distribution of workload over all sites, the actual objective function to be minimized is:

$$\sum_{e\in\mathcal{E}}(\alpha f_e^S + \beta \sum_{r\in\mathcal{R}} c_{er}^S) + M \max_{r\in\mathcal{R}} \sum_{j\in\mathcal{J}^S} l_j y_{jr}$$

In this last expression, M is a penalty factor, large enough to force the fair workload distribution in the solution without interfering with the network cost. Unless stated otherwise, the costs presented in this chapter were obtained for $\alpha = \beta = 1$.

5.3.1.2 Global Scenario

In order to dimension the network so it is capable of handling all individual scenarios, we must ensure that there is enough network capacity to handle each overload scenario (cf. section 5.2.3). We require therefore all constraints from the previous section for each possible source node $S \in \mathcal{R}$. Additionally, we introduce the following variables:

- $F_e =$ number of fibers on edge e for all scenarios,
- $C_{er} =$ average number of wavelengths departing from resource r carried by edge e over all individual scenarios, with
  $0 \leq \sum_{S\in\mathcal{R}} \sum_{j\in\mathcal{J}^S} \frac{\frac{d_j^I}{BT} + (|\mathcal{R}|-1)\frac{d_j^O}{BT}}{|\mathcal{R}|}$.

These variables adhere to:

$$\forall e \in \mathcal{E}, \forall S \in \mathcal{R}. F_e \geq f_e^S \tag{5.9}$$

$$\forall e \in \mathcal{E}, \forall r \in \mathcal{R}. C_{er} = \frac{\sum_{S\in\mathcal{R}} c_{er}^S}{|\mathcal{R}|} \tag{5.10}$$

The former constraint ensures sufficient fibers are activated to carry traffic for all scenarios, while the latter fixes the number of connections to the average over all scenarios. The network cost becomes in this case:

$$\sum_{e\in\mathcal{E}}(\alpha F_e + \beta \sum_{r\in\mathcal{R}} C_{er}) \tag{5.11}$$

Again, to enforce fair workload distribution, the use of a penalty factor may be necessary as explained in the previous section.

5.3.2 Parallelizing Heuristic

The previous section showed how to combine the individual scenarios into a model that is able to satisfy all individual scenarios at once. However, this approach becomes intractable very quickly for an increasing number of variables, in particular

because of the number of jobs. We therefore propose an alternative technique, which is able to return solutions within reasonable calculation time and resource limits, however at increased network cost.

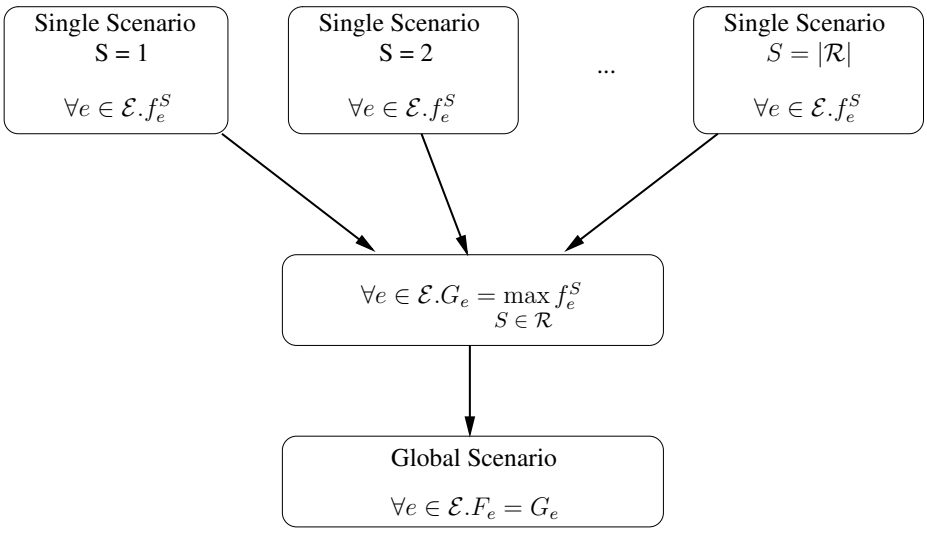


Figure 5.3: Parallelizing heuristic: overview

As illustrated in figure 5.3, we start by solving all individual scenarios independently. This step can be performed in parallel, and results in a series of fiber counts on each edge (variables f_e^S). These values are used to initialize the parameters G_e:

$$\forall e \in \mathcal{E}.G_e = \max_{S \in \mathcal{R}} f_e^S,$$

and then we proceed by solving the problem as defined in section 5.3.1.2, but replace constraint (5.9) by:

$$\forall e \in \mathcal{E}.F_e = G_e. \tag{5.12}$$

5.3.3 Incremental Heuristic

The parallelizing heuristic presented in section 5.3.2 performs a simple maximization over the solutions to a set of independently solved problems, instead of solving a single problem tackling all of these problems simultaneously.

Another heuristic method to solving this complex global problem is shown in figure 5.4. This *incremental* approach solves the dimensioning problem for a set of single excess load scenarios as follows. First, the elementary scenarios are ordered. The heuristic then solves the Grid network dimensioning problem for the

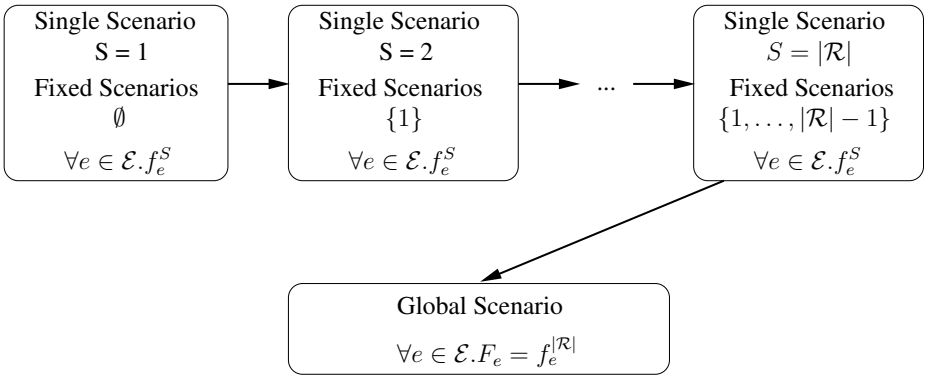


Figure 5.4: Incremental heuristic: overview

first elemental scenario in the ordered list and saves the resulting network dimensions, in particular the number of installed fibers on each edge and the number of activated wavelengths on each fiber in the scenario at hand.

Next, the heuristic solves a modified single scenario Grid dimensioning problem. The single scenario is the second scenario in the ordered list, while the modifications encompass the inclusion of the solution to the first dimensioning problem as additional constraints. Thus, in this second phase, the problem solved is to dimension the lambda Grid with minimal network cost, given that two scenarios need to be supported and that workload distribution and network routing for the first scenario have already been decided upon. The modifications thus add constraints to a standard single scenario dimensioning problem, but do not add additional variables.

This process is repeated; the number of iterations needed is the number of scenarios that needs to be supported. At iteration n, a modified single scenario dimensioning problem is solved for the nth scenario in the ordered list, given the solution to the previous $n - 1$ scenarios.

When the last iteration is finished, we have obtained a solution to the global dimensioning problem based upon this particular ordering of the elemental scenarios. By repeating this whole process for different scenario orderings, we can select the ordering with the lowest resulting lambda Grid dimensioning cost for the collection of all single source scenarios.

This incremental heuristic is able to produce better results when compared to the parallelizing heuristic, as will be demonstrated in section 5.4.10. As it is at the same time able to tackle large problems, it is our preferred solution method from section 5.5 onwards.

5.3.4 Equal Job Size Heuristic

Given workload L is generated by a large number of jobs n. In this case, the actual probability of a certain number of random jobs totalling a given workload depends on that number and the distributions associated with the arrival process and the job lengths. Therefore, we can approximate this set of jobs by substituting them by n jobs of equal length $\frac{L}{n}$. Assume that L_{max} is the a-priori maximum length of a single random job. Then typically, a large set of jobs totalling workload $L \ll nL_{max}$ will contain a relatively large number of "small" jobs (i.e., a length around $\frac{L}{n}$). This means that the total workload can be divided into $|\mathcal{R}| - 1$ parts of equal size plus some excess jobs of size $\frac{L}{n}$. Obviously, the case of n equal-sized jobs is a special instance of this. In this approximation, jobs are perfectly interchangeable.

Since we are handling a load balancing scenario, this implies that each resource must execute at least $\left\lfloor \frac{|\mathcal{J}^S|}{|\mathcal{R}|-1} \right\rfloor$ jobs. The assignment of the remaining jobs (at most $|\mathcal{R}| - 2$) is then limited by the fact that each resource may not receive more than one job (because of the load balancing constraint).

Clearly, it is not necessary to hold on to the binary variables y_{jr}^S for each job. Instead, we introduce new binary variables δ_r^S, which equal 1 iff one of the remaining jobs is executed on resource r, and 0 otherwise. Constraints (5.1) and (5.3) are replaced by

$$\sum_{r \in \mathcal{R}} \delta_r^S = |\mathcal{J}^S| \bmod (|\mathcal{R}| - 1), \tag{5.13}$$

while constraint (5.2) becomes

$$\delta_S^S = 0. \tag{5.14}$$

This effectively reduces the influence of the amount of excess jobs in the single scenario model, by elimination of $|\mathcal{J}^S||\mathcal{R}|$ job decision variables to $|\mathcal{R}|$, and $2|\mathcal{J}^S| + |\mathcal{R}|$ job-related constraints to only 2. The approximation presented here has been used in section 5.4.6 for large job count instead of the exact ILP formulation from section 5.3.1.

5.3.5 Divisible Load Theory

In the previous sections, integer linear programming formulations for the combined load distribution and the optical transport network dimensioning problem were presented. The central concept in these formulations (regarding the load distribution and thus traffic demand generation) is the use of per-job (integer) decision variables. These variables ensure that the workload distribution and network dimensioning (obtained by solving the ILP) is feasible for a given set of jobs.

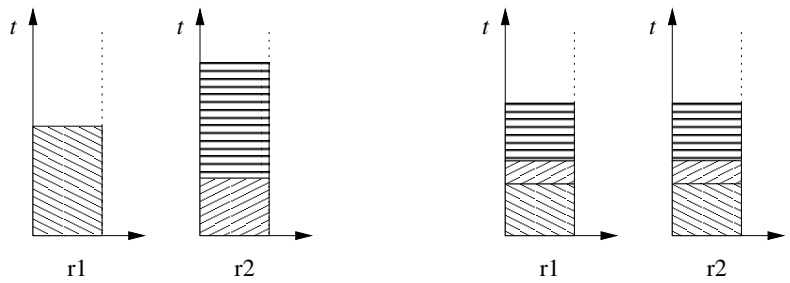


Figure 5.5: Sample schedule (3 jobs, 2 resources) when using the ILP method (left) and the DLT method (right)

However, as the number of jobs increases, the ILP's resulting complexity makes it difficult to obtain an optimal solution in reasonable time.

For steady state analysis of Grid systems processing large amounts of tasks, it has proven useful [14–16] to treat massively parallel applications as arbitrarily divisible. By extension, one can imagine treating the workload generated at a single Grid site as arbitrarily divisible. That is, one does not consider the individual jobs (each of which is, in reality, not divisible at all) but only takes into account the aggregate workload (i.e., sum-of-jobs) generated at each site during some interval T. Using the divisible load approach, the network dimensioning problem (and the related workload scheduling problem) can be restated as a linear programming problem without the per-job variables. The load distribution variables in this problem are now real-valued instead of integer.

As it is common for a workload to be described in terms of stochastic variables (e.g., interarrival time of the jobs, job length, etc.), we derived appropriate DLT parameters as described below.

5.3.6 Computational Resource Dimensioning

Our computational resource dimensioning strategy starts as follows. The necessary input to the Grid dimensioning problem includes the stochastic description of the jobs arriving at each site - more specifically, the distributions describing the job lengths and interarrival times. From these distributions, it is possible to calculate the $x\%$-percentile of the load arriving at each site per unit of time for varying values of x.

We chose to dimension each Grid site's computational resource in such a way that it is able to process a constant load per time unit equalling the $x\%$-percentile of the arriving workload, for a suitable value of x. An excess load scenario (as described in the previous section) then occurs when the actual workload arriving at a single site exceeds this value, and the excess load can be processed by the aggregation of the other sites.

5.3.7 Network Traffic Demand Derivation

As explained in section 5.2.2, jobs are submitted at their home site and it is assumed they find all needed input data on that site. In addition, in order to present job results to the end user, the generated output data is returned to that same home site. This job model implies that Grid network traffic is generated for a job if and only if that job is executed at a Grid site different from its home site.

In turn, using known distributions for job input and output data sizes (parameters d_j^I and d_j^O in section 5.2.2) in relation to the job's run time, the excess load and its distribution among remote computational resources leads to a probability distribution for the network traffic demand between Grid site pairs for the excess load scenario studied. Using a similar approach to the computational resource dimensioning, we can capture the data size per unit of computational work ratio using a suitable percentile value of this distribution.

For simple Grid OCS interconnection topologies in which wavelengths between sites are routed along a shortest path between these sites, the resulting network dimensioning cost for an excess load scenario can be analytically calculated as a function of the excess load and the used percentiles representing the arriving workload per time unit and the data size to workload ratio.

Indeed, for a Grid featuring N sites, let us make the following assumptions:

- let α_k^x denote the $x\%$-percentile of workload arriving at site k per unit of time, and thus, the capacity assigned to the computational resource located at site k
- let each fiber be limited to carrying W wavelengths, each able to transport data at a rate B
- let D represent the data size to workload ratio for job input as well as for job output data
- let site i be an overloaded site; that is, the actual workload arriving per time unit at site i is $\alpha_i^y > \alpha_i^x$

From these assumptions, it follows that the excess load arriving at site i can only be distributed among the other sites as long as $\alpha_i^y - \alpha_i^x \leq \sum_{k \neq i}(\alpha_k^x - \alpha_k^y)$. Furthermore, if fair distribution (i.e. proportional to each site's local processing capacity) of the excess load among the remote sites is desired it is required that $\forall k \neq i . \alpha_k^y \leq \alpha_k^x - (\alpha_i^y - \alpha_i^x)\frac{\alpha_k^x}{\sum_{l \neq i} \alpha_l^x}$. In case these conditions are met the bidirectional wavelength demand (i.e. the wavelength demands for input and output data, which are - under our assumptions - of equal size) between sites i and $k \neq i$ equals $\left\lceil D(\alpha_i^y - \alpha_i^x)\frac{\alpha_k^x}{B\sum_{l \neq i} \alpha_l^x} \right\rceil$.

If, for reasons of brevity, the demand from the excess load site i to a remote site k is abbreviated $d_k = \left\lceil D(\alpha_i^y - \alpha_i^x)\frac{\alpha_k^x}{B\sum_{l \neq i} \alpha_l^x} \right\rceil$, the network cost consisting of a

weighted sum of installed fibers and activated wavelengths in this single excess load scenario becomes for a full mesh network

$$\mathcal{C}_{mesh} = 2\sum_{k\neq i}\left(d_k + C\left\lceil\frac{d_k}{W}\right\rceil\right) \tag{5.15}$$

where W denotes the number of wavelengths per fiber as before and C represents the relative weight assigned to installed fibers when compared to the number of activated wavelengths. In other words, $C = \frac{\alpha}{\beta}$ where α and β are used as in the cost functions presented in section 5.3.1. The above formula then follows easily knowing that we have equal-sized wavelength demands in both directions and the number of fibers needed to carry d_k wavelengths is given by $\lceil\frac{d_k}{W}\rceil$. Similarly, for star and ring interconnection topologies, the resulting network cost becomes (by analyzing the link between the source node and the star point, and two half rings, respectively)

$$\mathcal{C}_{star} = C_{mesh} + 2\sum_{k\neq i} d_k + 2C\left\lceil\frac{\sum_{k\neq i} d_k}{W}\right\rceil \tag{5.16}$$

and

$$\begin{aligned}\mathcal{C}_{ring} = &\sum_{k=1}^{\lfloor\frac{N}{2}\rfloor}\left(\sum_{j=k}^{\lfloor\frac{N}{2}\rfloor} d_j + C\left\lceil\frac{\sum_{j=k}^{\lfloor\frac{N}{2}\rfloor} d_j}{W}\right\rceil\right)\\ &+\sum_{k=\lfloor\frac{N}{2}+1\rfloor}^{N-1}\left(\sum_{j=\lfloor\frac{N}{2}+1\rfloor}^{k} d_j + C\left\lceil\frac{\sum_{j=\lfloor\frac{N}{2}+1\rfloor}^{k} d_j}{W}\right\rceil\right)\end{aligned} \tag{5.17}$$

In expressions 5.15, 5.16 and 5.17, $\lceil x\rceil$ denotes the smallest integer larger than or equal to x, and $\lfloor x\rfloor$ denotes the largest integer smaller than or equal to x. The behavior of these cost functions (representing the optical network dimensioning cost obtained by studying a single excess load scenario on the given topology) has been visualized in figures 5.6, 5.7 and 5.8 for varying values of x and $y \geq x$. For the results in these figures, we have used the values $N = 13$, $W = 4$ and $C = 1$. The arriving divisible excess load is the load used throughout this chapter which is described in section 5.4.4. In figures 5.6, 5.7 and 5.8, it is assumed a single wavelength can carry 2.5 Gbps.

For the results in this chapter, however, we have not limited ourselves to these regular topologies, nor have we enforced shortest path routing. In these non-trivial cases, the required Grid dimensions and resulting network costs cannot be derived analytically and we are required to solve a linear program modeling the dimensioning problem. The objective of these programs is the minimization of the network cost, which takes on a similar structure as in expressions 5.15 through 5.17.

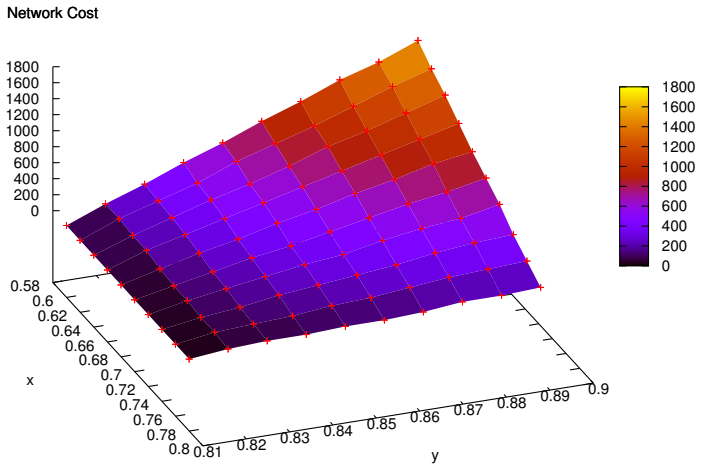


Figure 5.6: Network Dimensioning Cost: Mesh Topology

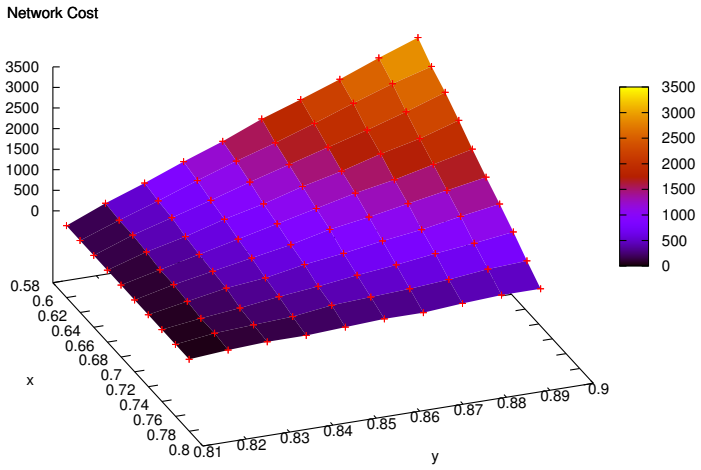


Figure 5.7: Network Dimensioning Cost: Star Topology

5.3.8 Lambda Grid Dimensioning Linear Program

The relevant portions of the linear program modeling the lambda Grid dimensioning problem in a scenario of a single overloaded computational resource r can now

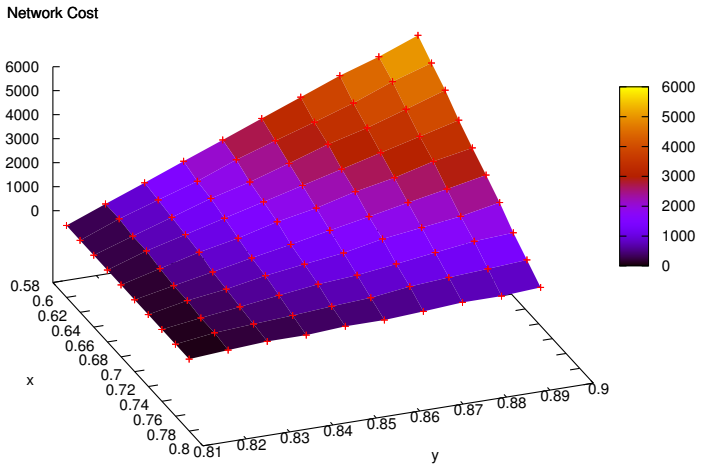


Figure 5.8: Network Dimensioning Cost: Ring Topology

be reformulated (using the DLT-derived parameters) as

$$\sum_{r \in \mathcal{R} \backslash \{S\}} \alpha_r = \alpha_S^y - \alpha_S^x \tag{5.18}$$

$$\forall r \in \mathcal{R} \backslash \{S\}. d_{Sr} \geq \frac{\alpha_r D_I}{B} \tag{5.19}$$

$$\forall r \in \mathcal{R} \backslash \{S\}. d_{rS} \geq \frac{\alpha_r D_O}{B} \tag{5.20}$$

$$\forall r \in \mathcal{R} \backslash \{S\}. \alpha_r + \alpha_r^y \leq \alpha_r^x \tag{5.21}$$

In these equations, α_r^y and α_r^x represent the y and x-percentiles for site r, calculated as described above. The amount of excess load (generated at resource S) that is scheduled for remote execution at resource r is dubbed α_r (real-valued). The value of D_I (D_O), which represents the average amount of input (output) data per processing unit, can be calculated from the job interarrival time, job length and job input (output) data size stochastic variables.

Most results presented in this chapter have been obtained in the case where $D_I = D_O = D$ and assuming that the excess load generated is distributed uniformly over the remote sites. If we examine the scenario where node 0 is the overloaded node, we then have that

$$d_{0r} = \left\lceil \frac{D(\alpha_0^y - \alpha_0^x)}{B(|\mathcal{R}| - 1)} \right\rceil \tag{5.22}$$

Under these assumptions, equations 5.15-5.17 become

$$\mathcal{C}_{mesh} = 2\sum_{r\neq 0}\left(d_{0r} + C\left\lceil\frac{d_{0r}}{W}\right\rceil\right), \tag{5.23}$$

$$\mathcal{C}_{star} = \mathcal{C}_{mesh} + 2\sum_{r\neq 0} d_{0r} + 2C\left\lceil\frac{\sum_{r\neq 0} d_{0r}}{W}\right\rceil \tag{5.24}$$

$$\begin{aligned}\mathcal{C}_{ring} = &\sum_{k=1}^{\lfloor\frac{|\mathcal{R}|}{2}\rfloor}\left(\sum_{r=k}^{\lfloor\frac{|\mathcal{R}|}{2}\rfloor} d_{0r} + C\left\lceil\frac{\sum_{r=k}^{\lfloor\frac{|\mathcal{R}|}{2}\rfloor} d_{0r}}{W}\right\rceil\right)\\ &+ \sum_{k=l}^{|\mathcal{R}|-1}\left(\sum_{r=l}^{k} d_{0r} + C\left\lceil\frac{\sum_{r=l}^{k} d_{0r}}{W}\right\rceil\right),\end{aligned} \tag{5.25}$$

where $l = \left\lfloor\frac{|\mathcal{R}|}{2} + 1\right\rfloor$.

5.4 Results and Discussion

5.4.1 ILP Solver

All integer linear programs needed to evaluate the different solution methods have been solved using ILOG CPLEX 8.0, running on an AMD Athlon XP1700+ based OpenMosix cluster (20 Debian GNU/Linux nodes) with 1 GB RAM per node.

5.4.2 Reference Topology

As a reference topology, we used the European core network depicted in figure 5.9, which is composed of 13 nodes and 17 bidirectional links. These links constitute fiber ducts; the exact number of fibers needed on each link follows from the solution to the dimensioning problem. As each OXC is located in a major European city, it is conceivable that each such cross connect has a Grid site attached to it. We therefore assume that each such OXC actually doubles as a Grid site (thus, we make abstraction of any access networks in place), so that our Grid has as many cross connects as Grid sites. This topology has a connectivity which resembles that of a bidirectional ring. Unless stated otherwise, we have solved the dimensioning problem on this European topology assuming each wavelength provides a data transfer rate of 2.5 Gbps, each fiber carries at most 4 wavelengths and the workload consisted of 2000 jobs instantiated as described in the following section.

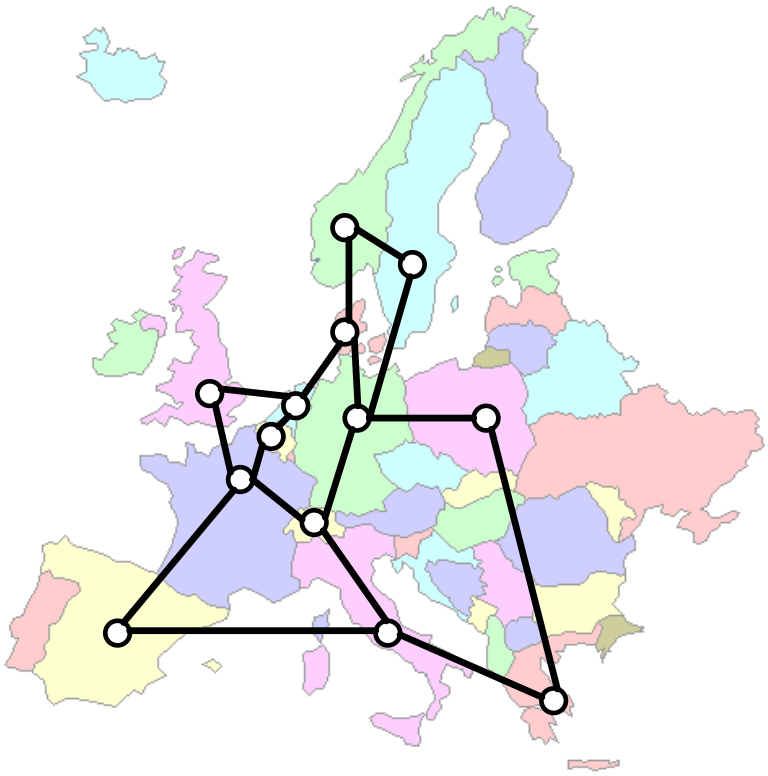


Figure 5.9: Reference Grid Topology: European Core Network (13 nodes, 17 bidirectional links)

5.4.3 Job Parameters

For the evaluation of the global optimization of the single overloaded source scenarios using an exact integer linear program, a heuristic decomposition of the exact program and the divisible load technique, we chose the following synthetic workload:

- job interarrival times are assumed to be independent and identically distributed, following an exponential distribution (thus, the number of jobs arriving over some interval follows a Poisson distribution),
- job lengths are also chosen to be independent and identically distributed, following a uniform distribution over $[0, L_{max}]$, with $L_{max} \ll T$,
- the size of the input data processed by a job is proportional to that job's length (factor D_I, resulting in size d_j^I - see also section 5.3.8),

- in analogy, the size of the output data generated by a job is proportional to that job's length (factor D_O, resulting in size d_j^O - see also section 5.3.8).

5.4.4 Excess Load

As previously explained, we assume a single computational resource is experiencing excessive load. We set the amount of load that can be handled locally (i.e., the computational resource's capacity) to be the 60%-percentile of the load as described by the arrival process and the job length distribution in the previous section. We assumed that an excessive load is made up by a job set instance whose aggregate load equals the 90%-percentile of the arriving workload. For our ILP model simulations featuring discrete jobs, we have generated 10 excess load job sets (2000 jobs each unless indicated otherwise) with total load equal to the difference of these percentiles. In particular, we have have kept the average interarrival time constant at $0.5s$. The average excess load per period of time is kept constant at 30 units of work per time unit by reducing L_{max} for increasing number of jobs per period. We have fixed the data size to computational workload ratio for these jobs while assuming $D_I = D_O$, and our choices have led to an average unidirectional excess load network demand of 220GBps.

Except for the results presented in section 5.4.10, it is always assumed that excess load (and thus, network demand) is distributed evenly among all remote sites (i.e. $k = |\mathcal{R}| - 1$).

5.4.5 Computational Complexity

We have compared the computational complexity of the (mixed) integer linear programs resulting from the application of the ILP, Heuristic and DLT methods in table 5.1, which lists the (order of magnitude of the) number of variables and constraints in each program as functions of network dimensions, number of resources and number of jobs.

As explained, the main simplification introduced by the DLT method is the absence of per-job variables, while the computational advantage of the heuristic over the ILP method lies in the reduced size of the individual subproblems. Note that e.g. the calculation of distribution percentiles for job length and data sizes is not contained in any of the complexity metrics in table 5.1.

The relation between the problem sizes listed in table 5.1 and the resulting run time on a single node of the cluster described in section 5.4.1 is listed in table 5.2, and clearly demonstrates the intractability of solving the pure ILP for large problem sizes.

Algorithm	Integer Vars	Float Vars	Constraints
ILP	$\|\mathcal{R}\|^2 \cdot (\|\mathcal{R}\| + \|\mathcal{E}\| + \|\mathcal{J}\|)$	0	$\|\mathcal{R}\| \cdot (\|\mathcal{R}\|^2 + \|\mathcal{E}\| + \|\mathcal{J}\|)$
Par. Heuristic	$\|\mathcal{R}\| \cdot (\|\mathcal{R}\| + \|\mathcal{E}\| + \|\mathcal{J}\|)$	0	$\|\mathcal{R}\|^2 + \|\mathcal{E}\| + \|\mathcal{J}\|$
Inc. Heuristic	$\|\mathcal{R}\| \cdot (\|\mathcal{R}\| + \|\mathcal{E}\| + \|\mathcal{J}\|)$	0	$\|\mathcal{R}\|^2 + \|\mathcal{E}\| + \|\mathcal{J}\| + (\|\mathcal{R}\| - 1) \cdot \|\mathcal{E}\|$
DLT	$\|\mathcal{R}\|^2 \cdot (\|\mathcal{R}\| + \|\mathcal{E}\|)$	$\|\mathcal{R}\| \cdot \|\mathcal{E}\|$	$\|\mathcal{R}\| \cdot (\|\mathcal{R}\|^2 + \|\mathcal{E}\|)$

Table 5.1: Algorithm Comparison: Computational Complexity (Reduction by factor $|\mathcal{R}|$ from ILP to parallelizing heuristic, and term $|\mathcal{J}|$ from ILP to DLT)

Algorithm	Time to 1st feasible solution	Time to good solution
ILP	≥ 24 hours	?
DLT	minutes	≈ 1 hour

Table 5.2: ILP-DLT Comparison: Computation Time on Single Cluster Node

5.4.6 ILP vs DLT

In figure 5.10, we have depicted the resulting cost for the dimensioning problem, applied to the topology from figure 5.9. These results show the cost for increasing number of jobs under the constraint that the total job load (per period) remains constant. Each value obtained for the parallelizing heuristic is the result of averaging the cost over ten different job instances. For low values of the job count (≤ 2000), we used the same approach for the costs obtained with the exact ILP formulation. The granularity of the job requirements causes the ILP method to perform better than the DLT approach in some cases, and worse in others. For higher number of jobs however, the computational intractability forced us to resort to the approximation described in section 5.3.4. In this case, each measurement is the average cost obtained from $|\mathcal{R}| - 2$ evaluations of this approximation for successive numbers of jobs around the measurement's corresponding x-value. Obviously, the cost of the DLT-based method remains constant as by its very nature, only the aggregate load is of importance. From the figure, it is clear that for high number of jobs (totalling a constant load), the cost for the ILP method converges to the DLT cost. Furthermore, the DLT-based approximation performs consistently better than the parallelizing heuristic, which reduces computational complexity by parallelizing the dimensioning problem into independent subproblems (maximal deviation from ILP method is about 0.5% in this case, vs. 5% deviation for the heuristic).

5.4.7 Connectivity

In the previous section, results were obtained for a single network topology. Figure 5.11 shows the resulting cost for the dimensioning problem when solved for a wider range of network topologies, for 2000 excess jobs per period. We have randomly generated connected networks (with number of nodes equal to the number of nodes in the reference network) for varying random-link probabilities p. Using

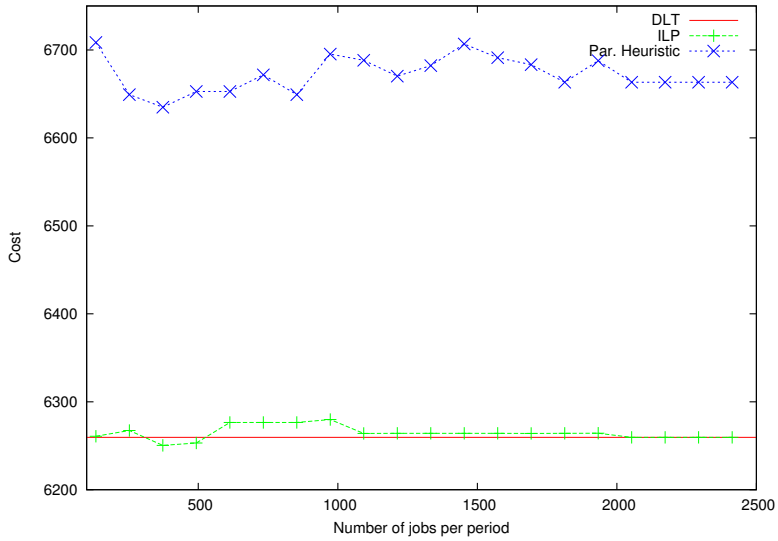


Figure 5.10: Cost vs. number of jobs per period for European network

this method, the European reference network is similar to the networks obtained for $p = 0.1$. For each value of p (except for $p = 1$, denoting a full mesh network), ten topologies have been generated. For all topologies, the DLT-based solution is very close to the solution obtained from the ILP, the difference staying below 1% for all values of p. Only in trivial cases ($p \to 0, p \to 1$) the parallelizing heuristic approach obtains the quality of the DLT-based solution.

5.4.8 Asymmetric Jobs

So far, symmetry between incoming and outgoing traffic for each job was assumed ($D_I = D_O$). Figure 5.12 shows the cost (for the reference network, again using 2000 excess jobs per period) for varying ratios of D_I to D_O but constant $D_I + D_O$. Because of input/output symmetry in our global dimensioning problem featuring input and output data sets of equal size, we expect these results to show symmetry around $s = \frac{D_I}{D_O} = 1$. In addition, due to the optimization over all individual scenarios, the chosen network dimensions are actually determined by $max(D_I, D_O)$. This is also an indicator that minimal cost is expected for $s = 1$. Clearly, the figure confirms these expectations.

5.4.9 Wavelength granularity

Results presented so far are obtained using at most 4 wavelengths per fiber, each wavelength able to carry 2.5 Gbps. Below, we evaluate the dimensioning prob-

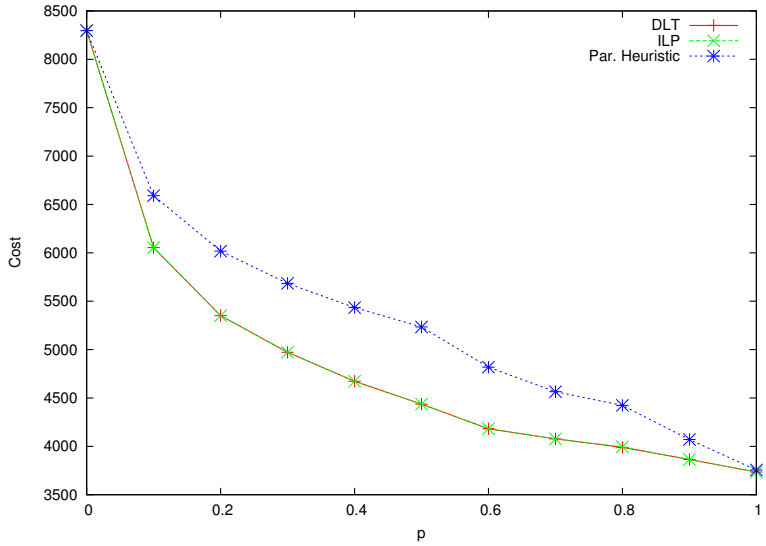


Figure 5.11: Cost vs. average connectivity for random networks with 13 nodes

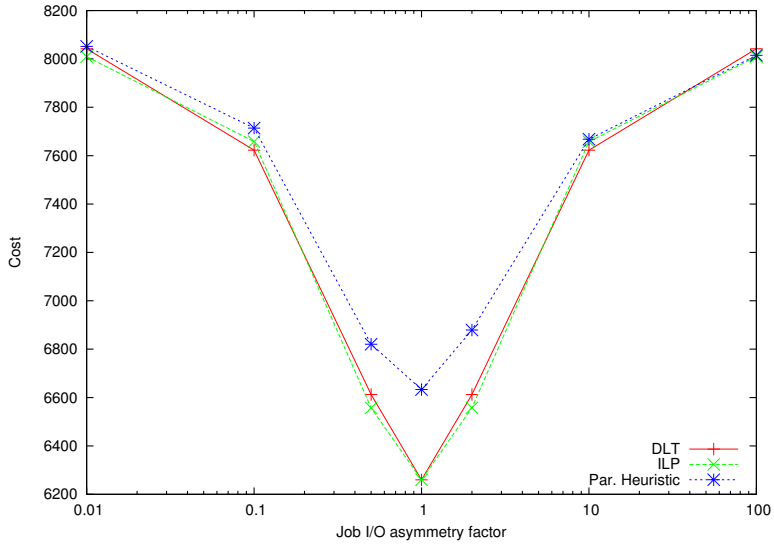


Figure 5.12: Cost vs. traffic asymmetry for European network

lem for the reference network for different wavelength granularities. In all cases, fiber capacity was fixed at 10 Gbps. This value and the wavelength granularity determine the number of wavelengths that can be carried on each fiber. Additionally, we compare different cost models of the wavelength per fiber parameter

$C = \frac{\alpha}{\beta}$. Each model is represented by a non-decreasing function, which mirrors the economic reality of the higher cost for technologies with larger wavelength capacity (figure 5.13). However, since smaller wavelength capacity implies a larger number of activated wavelengths, and thus increasing number of line termination equipment, we introduce three different functions for the parameter C. First, the constant function is invariant to changes in wavelength granularity. The second function scales the cost of a wavelength over a fiber linearly with the wavelength's bandwidth. Finally, three different geometric functions (factors 1.5, 2.5, and 3.5), bounded by the constant and linear functions, have been evaluated.

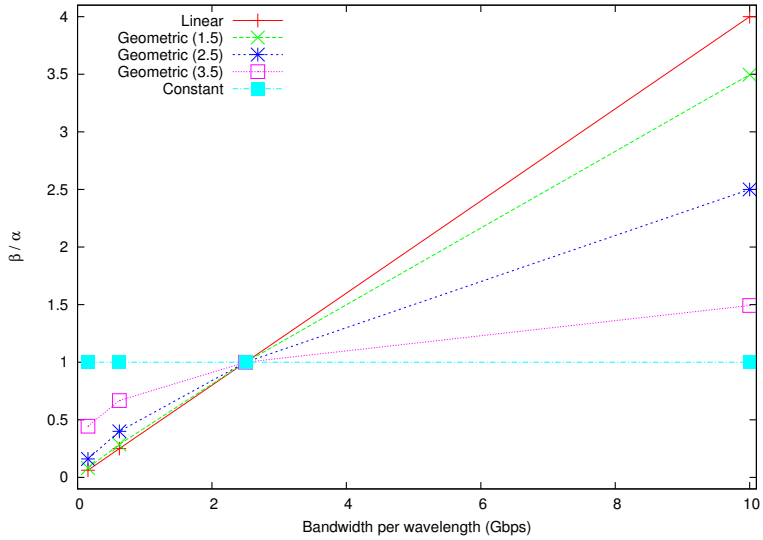


Figure 5.13: Different Wavelength/Fiber Cost Models vs. Wavelength Granularity

Table 5.3 summarizes our results for different wavelength bandwidths and cost models. For the reference case featuring $2.5Gbps$ wavelength granularity and $10Gbps$ fiber capacity, the cost is obtained as in equation 5.11 with $\alpha = \beta = 1$ - see also figure 5.13. For all other cases, α has been fixed at 1 (as fiber capacity remains fixed at $10Gbps$) while β (and thus, $\frac{\beta}{\alpha}$) takes on values as shown in figure 5.13. For all wavelength granularities presented (and thus, maximal number of wavelengths per fiber), our DLT-based approach outperforms the parallelizing heuristic and follows the ILP approach closely. This remains the case over all different cost models.

Figure 5.14 shows the resulting network cost obtained with the DLT approach for different models of the wavelength per fiber cost. Note that fiber capacity is fixed at 10Gbps in this figure, which explains the small cost differences for high wavelength data rates as the network cost is dominated by the fiber cost (which

B (Gbps)	Algorithm	Constant	Geo (1.5)	Geo (2.5)	Geo (3.5)	Linear
0.155	DLT	58361.1	27602.13	11815.0	7384.46	6241.2
	ILP	58241.08	27548.26	11795.06	7373.76	6254.76
	Par. Heuristic	58485.23	27755.44	12072.47	7739.88	6648.85
0.622	DLT	16780.2	12147.2	8407.4	6772.856	6241.2
	ILP	16760.0	12134.00	8398.68	6766.76	6254.96
	Par. Heuristic	16992.92	12395.82	8723.89	7136.62	6650.31
2.5	DLT	6259.54	6259.54	6259.54	6259.54	6259.54
	ILP	6259.54	6259.54	6259.54	6259.54	6259.54
	Par. Heuristic	6660.23	6660.23	6660.23	6660.23	6665.23
10	DLT	3605.77	4079.58	4986.93	5909.23	6364.6
	ILP	3605.77	4066.15	4986.92	5906.54	6364.62
	Par. Heuristic	4116.92	4555.38	5432.30	6309.23	6762.69

Table 5.3: Network cost for different wavelength/fiber cost models and wavelength granularity

grows bigger than the average wavelength cost) in these cases.

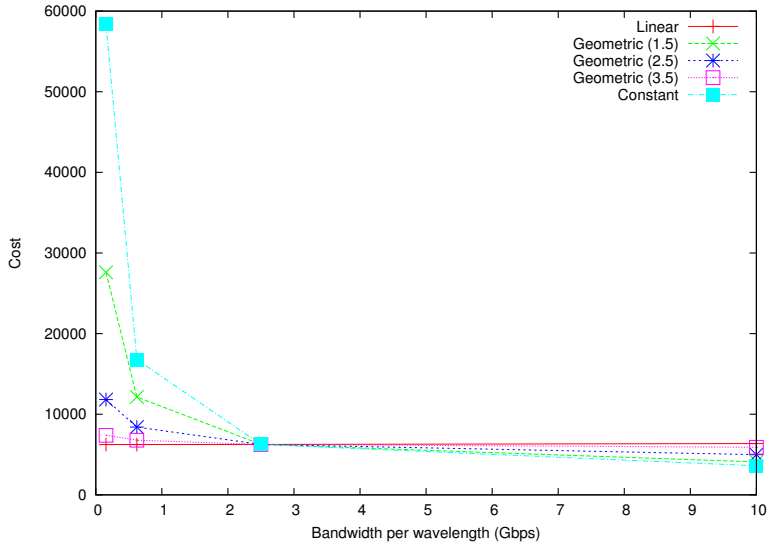


Figure 5.14: DLT Cost vs. Wavelength Granularity for European network under different wavelength/fiber cost models

5.4.10 Scheduling Strategies

In the previous sections, all results have been obtained for uniform excess load distribution over all remote sites (i.e., $k = 12$ for 13-node networks). As our model supports the combined dimensioning of the network and optimal selection

of load-balancing sites, figures 5.15 and 5.16 show the results for the DLT, ILP, parallelizing and incremental heuristic approaches for different scheduling strategies (k-values). These results were obtained on the previously used sets of randomly generated networks (10 networks per set) with average connectivity 0.1 and 0.9, respectively.

In all cases, the DLT and ILP approaches give similar network costs, outperforming both the parallelizing and incremental heuristics. Note though that in some cases, the DLT approach yields better solutions than the ILP approach, which can be attributed to the limited computational resources available to the ILP solver and enforced time constraints. The results for the parallelizing heuristic approach (as a function of k) can be explained as follows: for each individual excess load scenario, the heuristic "optimizes" network cost by selecting the k closest (with respect to hop count) remote sites. For high average connectivity values, the amount of sets resulting in minimal cost is higher than for lower average connectivity values. This means that the parallelizing heuristic approach (which does not correlate individual scenarios) is prone to selecting previously unused resources, resulting (after deciding on global capacities) in high network costs. This effect decreases for larger values of k, as the number of sets resulting in minimal network costs is lowered.

On the other hand, for low average connectivity values, the remote sites yielding minimal network cost are more likely to form a unique set. As such, higher k-values imply higher network costs due to the use of remote sites located further away (with regard to hop count). In addition, the difference between the network cost obtained through the heuristics and the network costs obtained by using the DLT or ILP approaches is smaller in the case of lower average connectivity because the heuristics are forced into using the same resource set as used by the DLT and ILP methods.

The same argument explains why the parallelizing heuristic seems to perform badly for high connectivity and low k-values when compared to the heuristics operating in the low connectivity, low k-value scenarios. This is because of the possible existence of multiple optimal solutions to a single overload scenario in the former case, which implies that a simple maximization (in case the parallelizing heuristic is used) over all these individual scenarios (without correlation) may perform erratically, depending on which optimal solution was selected in each individual scenario.

Better results are obtained using the incremental heuristic, as this heuristic offers the advantage that partial solutions are carried over to the next iteration. This subproblem correlation approach ultimately yields better results than the parallelizing heuristic. However, because even with the incremental heuristic the optimal solution to previously solved subproblems still remains fixed in all subsequent iterations, an optimal solution to the global problem is not necessarily obtained.

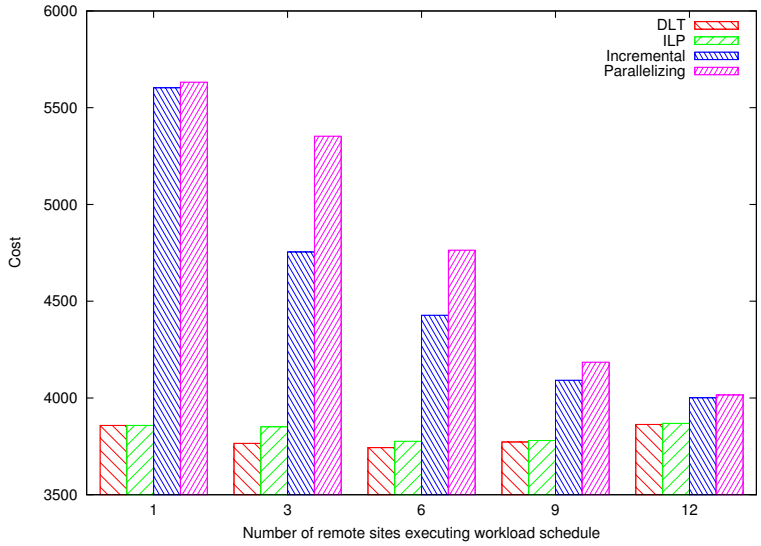


Figure 5.15: Cost vs. Scheduling Strategy for random networks with 13 nodes, $p = 0.9$

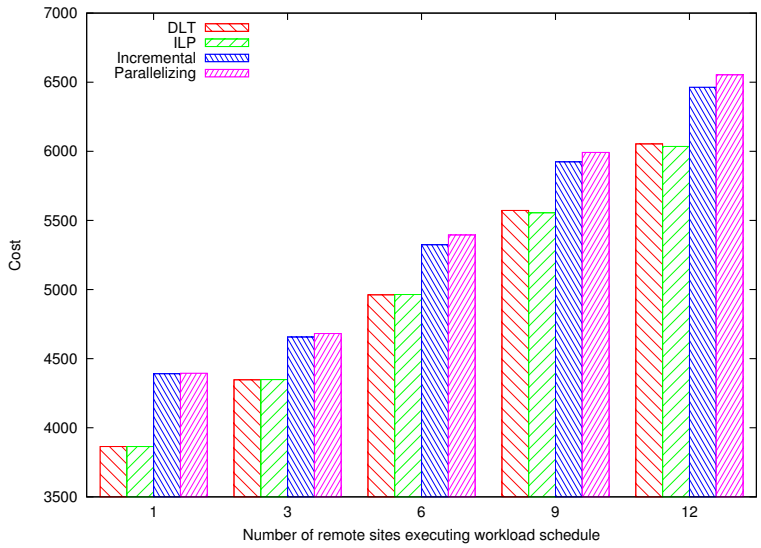


Figure 5.16: Cost vs. Scheduling Strategy for random networks with 13 nodes, $p = 0.1$

5.4.11 Applicability of DLT

The results presented so far show that the DLT approach to modeling and solving the optical transport network dimensioning problem (in the context of Grid excess

load handling) approximates the ILP based approach quite well in case this latter method becomes intractable due to large job numbers. This in turn means that the DLT approach can be used by network operators to obtain network costs for different interconnection topologies and compare them (e.g., in order to determine the most cost effective interconnection topology, given an excess load scenario that the resulting Grid must support). While we studied several parameter variations, so far we have limited our discussion to the base scenarios featuring a single excess load source and 100% dependable resources. In the next part of this chapter, we systematically study extensions to this base scenario along both axes.

5.5 Extension to Multiple Excess Load Sources

Up till now, the operational scenarios consisted of a single overloaded Grid site distributing its excess load to a collection of remote sites. In this section, we shed some light on the implications caused by considering Grid operational scenarios featuring multiple overloaded sites.

Assume that, in a Grid with N sites, each site's computational resource has a constant processing capacity of C. Furthermore, assume $k(1 \leq k \leq N-1)$ overloaded sites are present and the total aggregate load on these sites is fixed at $T \leq NC$. The amount of excess load generated at each of these k source sites is $\frac{T-kC}{k}$. Assuming that the excess load is distributed uniformly among the $N-k$ remote sites, let λ_k denote the total number of demanded wavelengths from all overloaded sites to each receiving remote site.

Using parameters D and B as in section 5.3.7, this quantity can be expressed as

$$\lambda_k = k \left\lceil \frac{L_k}{k} \right\rceil \tag{5.26}$$

where

$$L_k = D\frac{T-kC}{(N-k)B} \tag{5.27}$$

Expressing L_k in terms of L_1 yields

$$L_k = L_1 + \frac{D(k-1)(T-NC)}{B(N-1)(N-k)} \tag{5.28}$$

$$= L_1 - |\Delta_k| \tag{5.29}$$

Because of

$$\frac{|\Delta_{k+1}|}{|\Delta_k|} = \frac{k(N-k)}{(N-k-1)(k-1)} \tag{5.30}$$

$$\geq 1 \tag{5.31}$$

we have that $L_1 \geq L_2 \geq \ldots \geq L_{N-1}$. From equation 5.26, it follows that $\lambda_1 = \lceil L_1 \rceil \geq L_1$. We can now provide an upper bound on λ_k:

$$\lambda_k = k \left\lceil \frac{L_k}{k} \right\rceil \tag{5.32}$$

$$\leq L_k + k \tag{5.33}$$

$$= L_1 + k - |\Delta_k| \tag{5.34}$$

$$\leq \lambda_1 + k - |\Delta_k| \tag{5.35}$$

Since $T \leq NC$ we can distinguish between two cases:

- Case 1: $T = NC$. From the definition of $|\Delta_k|$ we can conclude that in this case $|\Delta_k| = 0$ and $\lambda_1 = \lambda_2 = \ldots = \lambda_{N-1}$.
- Case 2: $T < NC$. In this case, we have $|\Delta_k| > 0$.

Because k and λ_k are both integers, we can state that in both cases the following upper bound is valid for $2 \leq k \leq N-1$:

$$\lambda_k \leq \lambda_1 + k - 1 \tag{5.36}$$

The above results are valid if excess load from every source is distributed to all remaining remote sites. An alternative set of multi-source scenarios can be envisioned, using the concept of partitioning as explained in chapter 4. This way, the collection of remote sites is partitioned into k $(1 \leq k \leq \frac{N}{2})$ subsets. Each subset is dedicated to the absorption of excess load from one source only. If all remote sites are to be engaged in each scenario then in general there will be $k-1$ subsets containing $\left\lfloor \frac{N-k}{k} \right\rfloor$ remote sites and 1 subset with $\left\lceil \frac{N-k}{k} \right\rceil$ remote sites in it.

Assuming a Grid setup where $\frac{N-k}{k}$ is integer, and denoting the total number of wavelengths destined for a single remote site λ'_k (obviously, $\lambda'_1 = \lambda_1$), we have

$$\lambda'_k = \left\lceil \frac{L'_k}{k} \right\rceil \tag{5.37}$$

where

$$L'_k = D\frac{k(T-kC)}{(N-k)B} \tag{5.38}$$

$$= kL_k \tag{5.39}$$

Because $\lceil L_k \rceil \leq k \left\lceil \frac{L_k}{k} \right\rceil$, it follows that in this case

$$\lambda'_k \leq \lambda_k \tag{5.40}$$

Until now, the total aggregate load T on the collection of sources was assumed constant. Another set of multi-source scenarios consist of those scenarios where

the total excess load E generated by the source nodes is constant. For every k-source scenario, the total excess load is given by $E = T - kC$. Since schedulability requires that $T \leq NC$, this kind of scenario is only realistic for values of $k \leq \frac{NC-E}{C}$.

When excess load is distributed to all of the $N - k$ remote sites, we now have that

$$\lambda_k = k \left\lceil \frac{L_k}{k} \right\rceil \tag{5.41}$$

where

$$L_k = \frac{DE}{(N-k)B} \tag{5.42}$$

Because $k \leq \frac{NC-E}{C}$, L_k reaches its maximal value for $k = \frac{NC-E}{C}$. Since $\lambda_k \leq L_k + k$, it follows that

$$\lambda_k \leq L_k[k := \frac{NC-E}{C}] + k \tag{5.43}$$

$$= \frac{DC}{B} + k \tag{5.44}$$

If we again consider the approach where the Grid is partitioned into k subgrids, we can deduce the following bound:

$$\lambda'_k \leq \frac{DC}{B} + 1 \tag{5.45}$$

The above formulas bound the change in wavelength path demand when adding additional sources to the base scenario. The resulting lambda Grid dimensioning cost, however, also depends on the number of fibers required to support all elementary scenarios (see e.g. equation 5.11). To incorporate this factor some knowledge concerning the optical network's topology must be present.

For regular topologies with $k = 2$ sources and constant aggregate excess load, the exact effects on the cost components can be derived analytically. In a bidirectional ring OTN topology (one of the regular topologies discussed in section 5.3.8), for example, the base scenarios require a number of fibers on each of the $2N$ directed links equal to

$$\left\lceil \frac{\lfloor \frac{N}{2} \rfloor \lambda_1}{W} \right\rceil \tag{5.46}$$

as there exists a link in each scenario that needs to support traffic to $\lfloor \frac{N}{2} \rfloor$ remote sites. For the two-source partitioning scenarios, this figure is reduced to

$$\left\lceil \frac{\lceil \frac{N}{2} - 1 \rceil \lambda'_2}{W} \right\rceil \tag{5.47}$$

which is determined by observing the most loaded link in the ring in case the two source nodes are neighbors.

To calculate the average wavelength path cost, we continue as follows. Because of the topology's symmetry, we can limit our analysis to those scenarios where one source node is the node labeled 0. If the second source node is node S and shortest path routing is enforced, we can decide for each node k (different from 0 and S) which source node it should process excess load from. This is, because of shortest path routing, the closest source to node k i.e. either node 0 or node S, depending on

$$\min(d_{k,0}, d_{k,S}) \tag{5.48}$$

where

$$d_{k,0} = \begin{cases} k, & k \leq \left\lfloor \frac{N}{2} \right\rfloor \\ N-k, & k > \left\lfloor \frac{N}{2} \right\rfloor \end{cases} \tag{5.49}$$

and similarly

$$d_{k,S} = \begin{cases} (k-S) \mod N, & k = (S+1), \ldots, \left(S + \left\lfloor \frac{N}{2} \right\rfloor\right) \mod N \\ (S-k) \mod N, & k = \left(S + \left\lfloor \frac{N}{2} \right\rfloor + 1\right), \ldots, (S-1) \mod N \end{cases} \tag{5.50}$$

Assigning the closest excess load source to each site, the $N-2$ remote sites in the ring can be partitioned into 4 sets. Set S_1^S contains the nodes $1, 2, \ldots, \left\lfloor \frac{S}{2} \right\rfloor$, S_2^S contains $\left\lfloor \frac{N+S}{2} \right\rfloor + 1, \ldots, N-1$, set S_3^S consists of nodes $\left\lfloor \frac{S}{2} \right\rfloor + 1, \ldots, S-1$ and nodes $S+1, \ldots, \left\lfloor \frac{N+S}{2} \right\rfloor$ make up set S_4^S. Note that, depending on the exact value of $1 \leq S \leq N-1$, some of these sets may prove to be empty. For our analysis, we only need the *size* of these sets, given by

$$|S_1^S| = \begin{cases} 0, & S = 1 \\ \left\lfloor \frac{S}{2} \right\rfloor, & S > 1 \end{cases} \tag{5.51}$$

$$|S_2^S| = \begin{cases} 0, & S \geq N-2 \\ N-1-\left\lfloor \frac{N+S}{2} \right\rfloor, & S < N-2 \end{cases} \tag{5.52}$$

$$|S_3^S| = \begin{cases} 0, & S \leq 2 \\ S-1-\left\lfloor \frac{S}{2} \right\rfloor, & S > 2 \end{cases} \tag{5.53}$$

$$|S_4^S| = \begin{cases} 0, & S = N-1 \\ \left\lfloor \frac{N+S}{2} \right\rfloor - S, & S < N-1 \end{cases} \tag{5.54}$$

Using these numbers, the average wavelength path cost over all $\frac{N(N-1)}{2}$ scenarios in the bidirectional ring dual excess load source case can be written as

$$\frac{1}{N-1} \sum_{S=1}^{N-1} \sum_{i=1}^{4} \sum_{k=0}^{|S_i^S|} 2k\lambda_2' \tag{5.55}$$

equalling

$$\frac{\lambda_2'}{N-1}\sum_{S=1}^{N-1}\sum_{i=1}^{4}|S_i^S|(|S_i^S|+1) \tag{5.56}$$

In contrast, the average number of wavelength paths carried on the links in the base scenarios (where the links directly connected to the source node carry at worst traffic for $\lfloor \frac{N}{2} \rfloor$ nodes) is given by

$$2\lambda_1\left(\left\lfloor\frac{N}{2}\right\rfloor\right)^2 \tag{5.57}$$

Comparing the costs of the dual source partitioning scenario to the base scenario on the bidirectional ring thus yields (for $C = 1$)

$$\frac{\frac{\lambda_2'}{N-1}\sum_{S=1}^{N-1}\sum_{i=1}^{4}|S_i^S|(|S_i^S|+1)+2N\left\lceil\frac{\lceil\frac{N}{2}-1\rceil\lambda_2'}{W}\right\rceil}{2\lambda_1\left(\lfloor\frac{N}{2}\rfloor\right)^2+2N\left\lceil\frac{\lfloor\frac{N}{2}\rfloor\lambda_1}{W}\right\rceil} \tag{5.58}$$

For our reference parameter choices ($C = 1$, $W = 4$, $N = 13$, wavelength granularity $2.5Gbps$ and excess load generated as in section 5.4.4) this fraction equals 0.9.

If we consider a full mesh topology instead, the average wavelength path cost changes from

$$2(N-1)\lambda_1 \tag{5.59}$$

in the base scenario to

$$2(N-2)\lambda_2' \tag{5.60}$$

when two-source partitioning scenarios are being considered.

Each link now requires

$$\left\lceil\frac{\lambda_2'}{W}\right\rceil \tag{5.61}$$

fibers, compared to the

$$\left\lceil\frac{\lambda_1}{W}\right\rceil \tag{5.62}$$

fibers required per link in the base scenario.

The change in cost incurred by the two-source partitioning scenario on a full mesh topology is thus (for $C = 1$)

$$\frac{2(N-2)\lambda_2'+N(N-1)\left\lceil\frac{\lambda_2'}{W}\right\rceil}{2(N-1)\lambda_1+N(N-1)\left\lceil\frac{\lambda_1}{W}\right\rceil} \tag{5.63}$$

which yields 1.03 for the reference parameter settings we used.

In figure 5.17, we have plotted the dimensioning cost for the random network dual excess load source scenario problem and compared it to the dimensioning cost in the base scenario. The excess load is kept constant and has been generated following the recipe described in section 5.4.4. Again, for each value of p, 10 networks have been taken into account. Note how the cost differences shown in this figure are in line with the values predicted by equations 5.58 and 5.63, roughly corresponding to the cases $p = 0.1$ and $p = 1$, respectively. As p increases, more routing opportunities are available in the networks, decreasing the possibility of badly (i.e. resulting in high network costs) situated sources, as in the bidirectional ring scenario featuring two neighboring sources. Thus, for high values of p there is not much to be gained when half of the excess load is generated at a second "well-placed" source. On the contrary, as the same excess load is now processed by $|\mathcal{R}| - 2$ remote sites, the cost for the dual source scenario may surpass the cost of the base scenarios for high p values, as indicated by the numerical evaluation of expression 5.63.

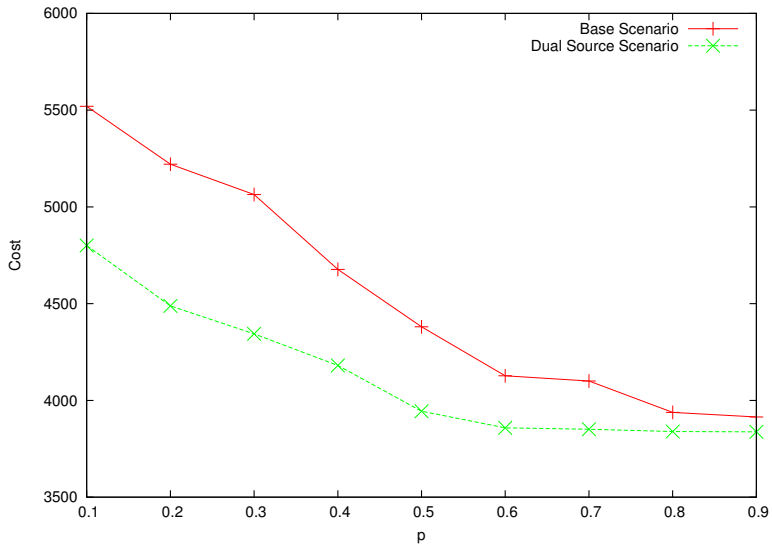


Figure 5.17: Dual Source Scenario Cost for Random Networks

5.6 Extension to Resource Failure Scenarios

In the initial set of scenarios we have studied only involved a single source node and assumed reliable Grid resources. In the previous section, we have extended our notion of scenario to include multiple source nodes, still assuming these resources do not fail.

In this section, we extend our base scenarios into another direction, taking into account possible resource failures. In particular, a single elementary scenario in this section is described by a (source,failure) pair, denoting the single source present in each scenario, and the single resource failure (if any) occurring in this scenario.

We can discern 3 kinds of resource failures, corresponding to the most prominent resource types in our analysis:

- computational resource failures
- optical cross-connect failures
- network link failure, effectively cutting all fibers inside

The effects of a computational resource failure are that the affected resource cannot handle any excess load from the overloaded source. The wavelength routing capabilities of the underlying optical cross-connect, however, are deemed to remain operational. In contrast, in case of an optical cross-connect failure, network traffic can no longer be routed over this cross-connect. This means the failure scenario can be treated as a base scenario where the affected cross-connect and all links incident to and from it have been left out of the original network. The last failure scenario only affects a single link.

Note that an optical cross-connect failure implies the unavailability of the connected computational resource and thus encompasses the computational resource failure scenario. In addition, the unavailability of an optical cross-connect implies that the links connected to it remain unused (it is assumed that no jobs are submitted to the OXC's corresponding computational resource, as there is effectively no way to route these jobs elsewhere).

Therefore, when representing the cost of protecting the lambda Grid against different types of resource failures, we will limit ourselves to the additional network dimensioning cost related to the protection against optical cross-connect failures and compare this cost to the cost associated with the single source scenarios discussed previously in this chapter.

Thus, the failure scenarios for which numerical results have been obtained are the optical cross connect failure scenarios, under the condition that such a failure does not partition the Grid's interconnecting network.

To accommodate our new definition of the elementary scenario (encompassing both a source node and failing node), the combined dimensioning and workload scheduling linear program needs to be adapted as follows.

5.6.1 Computational Resource Failure

The notion of a single failing computational resource at node n can easily be incorporated into the combined dimensioning and workload scheduling linear program

by modifying equation 5.18 to read

$$\sum_{r \in \mathcal{R} \backslash \{S,n\}} \alpha_r = \alpha_S^y - \alpha_S^x \tag{5.64}$$

This effectively excludes any excess load generated at source node S from being scheduled on the now-defunct computational resource at node n.

5.6.2 Optical Cross-Connect Failure

To model the failure of an optical cross-connect, note that such an optical cross-connect will be completely unused if all links incident to and from it are void of traffic. Therefore, we can model the failure of an optical cross-connect at node n by modifying network flow equation 5.6 to exclude these links from the flow conservation constraints as follows:

$$\forall u \in \mathcal{R} \backslash \{n\}, \forall v \in \mathcal{R} \backslash \{u,n\}. \sum_{e \in \mathcal{E}_v^+ \backslash \mathcal{E}_n^-} c_{eu}^S + d_{uv}^S = \sum_{e \in \mathcal{E}_v^- \backslash \mathcal{E}_n^+} c_{eu}^S \tag{5.65}$$

Indeed, as network cost increases when more fibers and wavelengths are activated and we are dealing with a cost minimization problem, the net effect of the exclusion is that the affected cross-connect as well as any links to and from it will not be used in the observed scenario. After all, suppose that a solution to the network dimensioning problem is obtained in which an affected link is not void of traffic, we can immediately derive another valid solution which is cheaper.

Note that only those failure scenarios preserving the network's connectedness have been studied, as stated in section 5.6.

5.6.3 Link Failure

In a similar way to the approach described in the previous section, we can model the failure of a single link l by modifying flow equation 5.6. Replacing that equation with

$$\forall u \in \mathcal{R} \backslash \{n\}, \forall v \in \mathcal{R} \backslash \{u,n\}. \sum_{e \in \mathcal{E}_v^+ \backslash \{l\}} c_{eu}^S + d_{uv}^S = \sum_{e \in \mathcal{E}_v^- \backslash \{l\}} c_{eu}^S \tag{5.66}$$

ensures that traffic is routed over operational links only. Note that link failure in a network topology featuring low average connectivity may result in a disconnected network.

5.6.4 Impact on Dimensioning Cost

In this section, we study the increased dimensioning cost incurred by considering the possible optical cross-connect failure scenarios and compare it to the dimensioning cost of the base scenario featuring a single source and no failures.

From our new definition of the elementary scenario, it follows that the number of possible scenarios greatly increases when adding the concept of single resource failures. Therefore, from this point on, all dimensioning costs in this chapter have been obtained using the incremental heuristic described in section 5.3.3. As this heuristic evaluates the scenarios sequentially, we have repeated each heuristic run 10 times (for different scenario orderings) as to reduce the resulting solution's sensitivity to scenario reordering, as demonstrated by the numbers shown in figure 5.18 which represent sequential improvements in network dimensioning cost when dimensioning our set of 13-node random networks (with $p = 0.1$) for the optical cross-connect failure scenarios using the incremental heuristic. Again, the excess load used in these scenarios is the one described in section 5.4.4.

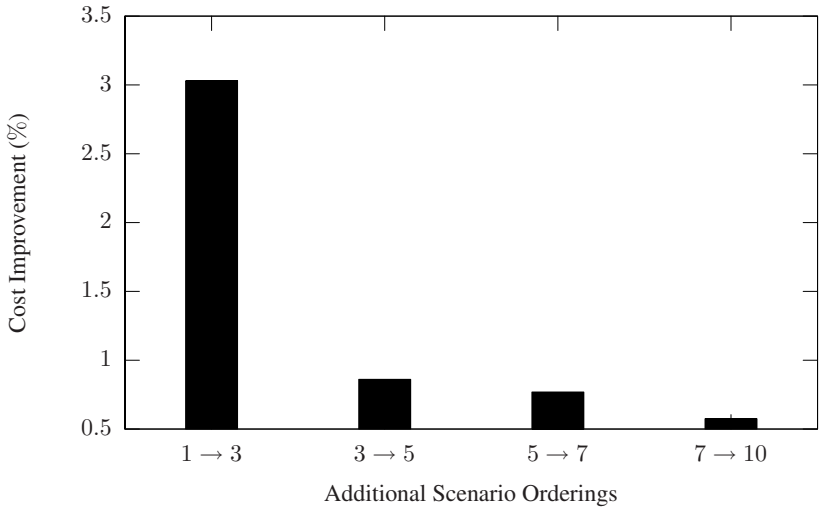


Figure 5.18: Incremental Heuristic: Sensitivity to Number of Investigated Scenario Orderings

We have performed the lambda Grid dimensioning for the failure scenarios for different parameter sets including the Grid's scheduling strategy, the wavelength granularity and job I/O asymmetry.

Again, for the reference case we assume uniform workload distribution (featuring perfect I/O symmetry) over all operational remote nodes and assume $2.5Gbps$ wavelengths. The cost increase for this reference case on our set of random networks due to OXC failure protection has been plotted in figure 5.19.

For a regular topology like the bidirectional ring, it is possible to derive analytical results with regard to the expected additional costs components (both wavelength path and fiber costs) as follows.

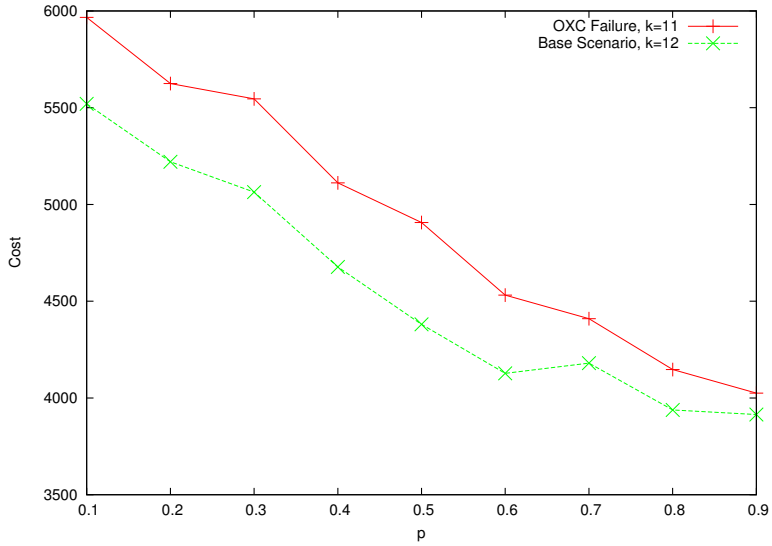


Figure 5.19: OXC Failure Protection Cost Increase for Random Networks

If the excess source node is node 0 and the OXC at node $n-1$ fails, the link carrying the bulk of the traffic is situated between nodes 0 and 1, as all working traffic for nodes $1, \ldots, \lfloor \frac{N}{2} \rfloor$ as well as traffic routed on the backup path for nodes $\lfloor \frac{N}{2} \rfloor + 1, \ldots, N-2$ is routed over this link.

In order to support all node failure scenarios in such a bidirectional ring, each of the $2N$ directed links needs to be provisioned with at least an amount of fibers equal to

$$\left\lceil \frac{\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right)(N-2)}{W} \right\rceil \tag{5.67}$$

This number is obtained by observing the link between nodes 0 and 1 and assuming that redistributing λ_1 wavelengths from the failing node to the $N-2$ remaining ones increases the wavelength demand for each of those nodes to $\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil$. In contrast, in the single site excess load scenario (without taking into account possible OXC failures) this number is only

$$\left\lceil \frac{\lambda_1 \lfloor \frac{N}{2} \rfloor}{W} \right\rceil \tag{5.68}$$

In the envisioned bidirectional ring failure scenario, the number of wavelength paths (originating from node 0, with node F failing) carried on the network links

equals

$$2\sum_{k=1}^{F-1} k(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil) + 2\sum_{k=F+1}^{N-1} (N-k)(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil) \tag{5.69}$$

Thus, averaged over all scenarios we obtain

$$\frac{\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil}{N-1} \sum_{F=1}^{N-1} \left(2\sum_{k=1}^{F-1} k + 2\sum_{k=F+1}^{N-1} (N-k) \right) \tag{5.70}$$

which reduces to

$$\frac{2N(N-2)\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right)}{3} \tag{5.71}$$

For the base scenarios, this average number of wavelength paths carried on the links is given by

$$2\sum_{k=1}^{\left\lfloor \frac{N}{2} \right\rfloor} k\lambda_1 + 2\sum_{k=\left\lfloor \frac{N}{2} \right\rfloor + 1}^{N-1} \left(k - \left\lfloor \frac{N}{2} \right\rfloor \right) \lambda_1 \tag{5.72}$$

which in turn can be simplified to

$$2\lambda_1 \left(\left\lfloor \frac{N}{2} \right\rfloor \right)^2 \tag{5.73}$$

For a bidirectional ring topology, the cost increase for the OXC failure protection (compared to the base scenario) is therefore (in case $C = 1$)

$$\frac{\frac{2N(N-2)\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right)}{3} + 2N \left\lceil \frac{\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right)(N-2)}{W} \right\rceil}{2\lambda_1 \left(\left\lfloor \frac{N}{2} \right\rfloor \right)^2 + 2N \left\lceil \frac{\lambda_1 \left\lfloor \frac{N}{2} \right\rfloor}{W} \right\rceil} \tag{5.74}$$

which equals 1.60 under our set of assumptions concerning shortest-path routing and equal-sized input and output data sets, using excess load as described in section 5.4.4 and the values $C = 1$, $N = 13$, $W = 4$ and wavelength granularity $2.5Gbps$.

From figure 5.19, it follows that in our reference case the cost increase incurred by providing OXC failure resilience to the base scenarios is no more than 10% for our set of random networks.

The figures obtained for the bidirectional ring topology are much higher because that particular topology features a failure scenario in which all traffic is rerouted over a single network link (regardless of the excess load source under

observation), thus making the bidirectional ring the worst envisionable topology with low average connectivity (while being resilient to single node failures).

Furthermore, in the above formulas it has been assumed that network traffic in the base scenario fills exactly λ_1 wavelength paths.

For a full mesh topology in which shortest-path routing is enforced, the average number of wavelength paths on the links in over all OXC failure scenarios changes to

$$2(N-2)\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right) \tag{5.75}$$

from

$$2(N-1)\lambda_1 \tag{5.76}$$

The total number of fibers needed over all scenarios is then given by

$$N(N-1)\left\lceil \frac{\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right)}{W} \right\rceil \tag{5.77}$$

and is simply given by

$$N(N-1)\left\lceil \frac{\lambda_1}{W} \right\rceil \tag{5.78}$$

in absence of OXC failures.

Under our assumptions, the relative cost increase due to OXC failure protection for full mesh interconnection networks is thus given by (again, $C = 1$)

$$\frac{2(N-2)\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right) + N(N-1)\left\lceil \frac{\left(\lambda_1 + \left\lceil \frac{\lambda_1}{N-2} \right\rceil\right)}{W} \right\rceil}{2(N-1)\lambda_1 + N(N-1)\left\lceil \frac{\lambda_1}{W} \right\rceil} \tag{5.79}$$

which yields 1.06 with our reference parameter settings. Note how this value corresponds closely to the cost difference shown in figure 5.19 for highly connected networks ($p = 0.9$). As for higher p values more routing opportunities are available in the networks, protection from single OXC failures comes at lower additional cost when compared to the networks corresponding to lower values of p. Of course, the ultimate network dimensioning cost when protecting against single OXC failures is higher than the costs for the base scenarios, as excess load is now distributed among only $|\mathcal{R}| - 2$ remote sites.

5.6.4.1 Job I/O Asymmetry

While in the reference case jobs are assumed to produce as much output data as they need input data, figure 5.20 shows the dimensioning costs for OXC failure resilient lambda Grids for I/O asymmetric jobs. These results are obtained for our

set of 10 random networks corresponding to $p = 0.1$ and excess load generated following the recipe described in section 5.4.4.

The reasons for the existence of the symmetry and local minimum have been explained in section 5.4.8. For all asymmetry factors studied, the additional dimensioning cost in case OXC failure protection is incorporated does not exceed 10%. The cost increase is maximal around the point where input and output data are of equal size (i.e. $D_I = D_O$, see section 5.4.8). In this case, network dimensions are determined by $\max(D_I, D_O)$ which is minimal when $D_I = D_O$ (if $D_I + D_O$ is constant). As taking into account possible OXC failures needs extra network capacity when compared to the base scenario, the network capacity needed to support possible OXC failures will differ most from the base scenario cost in case $D_I = D_O$ as in this case there is minimal room for wavelength and fiber re-use (and thus, the most extra capacity needs to be installed when $D_I = D_O$).

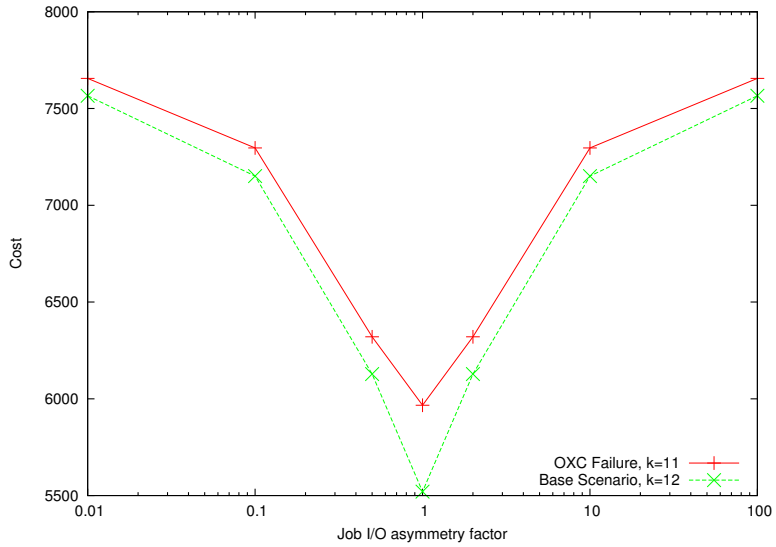


Figure 5.20: Traffic Asymmetry: OXC Failure Protection Cost for Random Networks ($p = 0.1$)

5.6.4.2 Wavelength Granularity

As in section 5.4.9, we have revisited the resulting network dimensioning costs for different wavelength granularities ($155Mbps$, $622Mbps$, $2.5Gbps$ and $10Gbps$) while limiting the number of wavelengths per fiber in such a way that total fiber capacity remains fixed at $10Gbps$. Again, the results are obtained for our set of 10 random networks corresponding to $p = 0.1$ and excess load generated as described in section 5.4.4. We used the linear wavelength/fiber cost model discussed

in section 5.4.9 - to compare to the values in table 5.3, results in figure 5.21 must be divided by the correct value of β. For all wavelength granularities examined, the dimensioning cost for the OXC failure resilient lambda Grid does not exceed the base scenario dimensioning cost by more than 5%, as is demonstrated in figure 5.21.

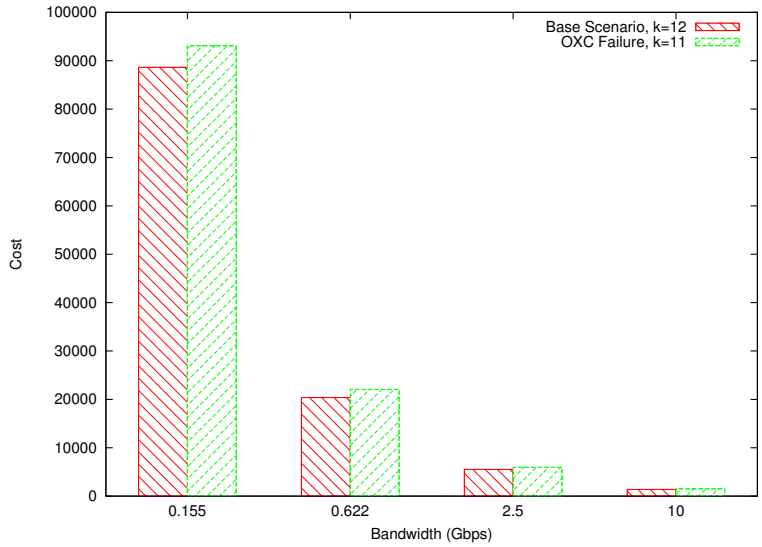


Figure 5.21: Wavelength Granularity: OXC Failure Protection Cost for Random Networks ($p = 0.1$)

5.6.4.3 Scheduling Strategies

In the previous sections, excess workload in each scenario was distributed among all remote Grid sites. Figure 5.22 compares the resulting dimensioning cost for the base scenarios and the OXC failure resilient lambda Grid for varying numbers of remote sites participating in the excess workload absorption. For this figure, we used the same set of 10 random networks (for $p = 0.1$) again, as well as the excess load from section 5.4.4. We see comparable increases (between 5 and 10%) in dimensioning cost for all numbers of active remote sites.

5.7 Conclusions

In this chapter, we have studied the dimensioning problem of an optical circuit switched transport network for Grid applications. The initial operational scenario considered was that of a single Grid site generating excess load. We presented

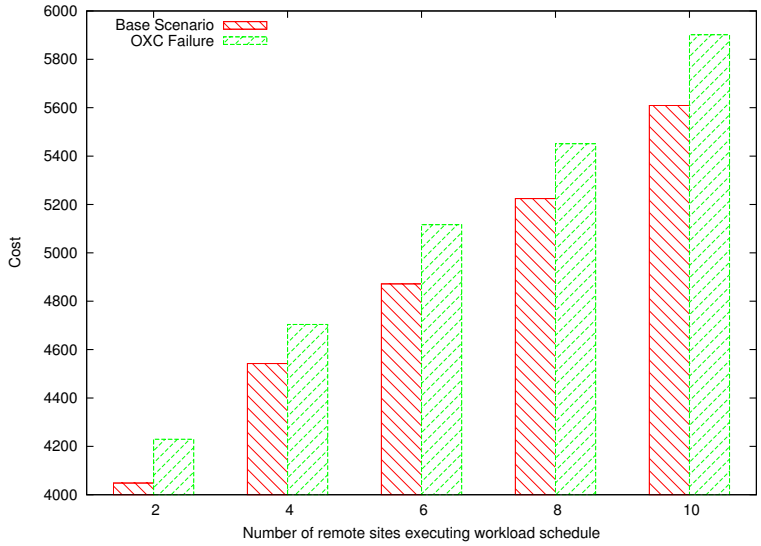


Figure 5.22: Scheduling Strategy: OXC Failure Protection Cost for Random Networks ($p = 0.1$)

and discussed a solution for this dimensioning problem using a model based on divisible load theory (DLT). We compared this model to an integer linear programming formulation using an exact job-level workload description, and proposed additional simplifications to solve the problem. These simplifications consist of a parallelizing heuristic and an incremental heuristic, both attempting to solve the resulting linear programs in a more timely fashion.

Results show that the global optimization of single overloaded source scenarios using the exact job-level ILP formulation is possible only for a low number of jobs. However, we have established the convergence of the DLT-based approach and this job-level ILP formulation for increasing number of jobs. This indicates that the DLT formulation is of practical use in cases where the exact ILP becomes computationally intractable. Additionally, we have presented heuristic methods based on the job-level ILP model. These heuristics show better scaling behavior for high number of jobs, although they are consistently outperformed by the DLT-based method.

We validated these conclusions for a wide range of parameter variations, most notably network topology (through variation in average link probability), wavelength granularity and cost model, changes in traffic demand (a)symmetry and Grid scheduling policy.

In addition, we elaborated on additional types of scenarios, including scenarios featuring multiple Grid excess load sites and scenarios demonstrating resilience

against single resource failures. For the topologies and scenarios studied in this chapter, the additional lambda Grid dimensioning cost incurred by explicitly incorporating possible optical cross-connect failures in the dimensioning problem remained below 10% when compared to the dimensioning cost of our base problem.

We can conclude that our DLT-based approach is of practical use to network operators interested in selecting and dimensioning a suitable OCS Grid interconnection topology, including selection of optimal wavelength granularity.

References

[1] *The TeraGrid project*. `http://www.teragrid.org/`.

[2] D. Simeonidou, R. Nejabati, B. St. Arnaud, M. Beck, P. Clarke, D. B. Hoang, D. Hutchison, G. Karmous-Edwards, T. Lavian, J. Leigh, J. Mambretti, V. Sander, J. Strand, and F. Travostino. *Optical Network Infrastructure for Grid*. Global Grid Forum Informational Document, 2004.

[3] L. Smarr, A. Chien, T. DeFanti, J. Leigh, and P. Papadopoulos. *The OptIPuter*. Communications of the ACM, 46:58–67, 2003.

[4] T. DeFanti, C. de Laat, J. Mambretti, K. Neggers, and B. Arnaud. *TransLight: A Global-Scale LambdaGrid for e-Science*. Communications of the ACM, 46:34–41, 2003.

[5] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. *Theory and Practice in Parallel Job Scheduling*. In Dror G. Feitelson and Larry Rudolph, editors, Job Scheduling Strategies for Parallel Processing, pages 1–34. Springer Verlag, 1997.

[6] L. Hall, A. Schulz, D. Shmoys, and J. Wein. *Scheduling To Minimize Average Completion Time: Off-line and On-line Algorithms*. In SODA: ACM-SIAM Symposium on Discrete Algorithms (Conference on Theoretical and Experimental Analysis of Discrete Algorithms), pages 142–151, 1996.

[7] L. Hall, A. Schulz, D. Shmoys, and J. Wein. *Scheduling To Minimize Average Completion Time: Off-line and On-line Algorithms*. In SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), pages 142–151, 1996.

[8] M. Hovestadt, O. Kao, A. Keller, and A. Streit. *Scheduling in HPC Resource Management Systems: Queueing vs. Planning*. In Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2862/2003, pages 1–20, 2003.

[9] J.S. Choi. *A Survey of Routing and Wavelength Assignment Schemes in Wavelength Routed Optical Networks: Static Case*. In Proceedings of Wireless and Optical Communications (WOC2003), pages 91–108, 2003.

[10] N. Wauters and P. Demeester. *Design of the Optical Path Layer in Multiwavelength Cross-Connected Networks*. IEEE Journal on Selected Areas in Communications, 14:881–892, 1996.

[11] D. Banerjee and B. Mukherjee. *Wavelength-Routed Optical Networks: Linear Formulation, Resource Budgeting Tradeoffs, and a Reconfiguration Study*. IEEE/ACM Transactions on Networking, 8:598–607, 2000.

[12] D. Coudert and H. Rivano. *Lightpath Assignment for Multifibers WDM Networks with Wavelength Translators*. In Proceedings of IEEE Globecom'02, volume 3, pages 2686–2690, 2002.

[13] M. Tornatore, G. Maier, and A. Pattavina. *WDM network optimization by ILP based on source formulation*. In Proceedings of IEEE Infocom'02, volume 3, pages 1813–1821, 2002.

[14] D. Yu and T.G. Robertazzi. *Divisible Load Scheduling for Grid Computing*. In Proceedings of the IASTED 2003 International Conference on Parallel and Distributed Computing and Systems (PDCS), 2003.

[15] J.T. Hung, H.J. Kim, and T.G. Robertazzi. *Scalable Scheduling in Parallel Processors*. In Proceedings of the 36th Annual Conference on Information Sciences and Systems (CISS'02), pages 20–22, 2002.

[16] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. *A realistic network/application model for scheduling divisible loads on large-scale platforms*. Rapport de recherche de l'INRIA-Rhone-Alpes (RR-5197), 2004.

[17] P. Thysebaert, M. De Leenheer, B. Volckaert, F. De Turck, B. Dhoedt, and P. Demeester. *Scalable dimensioning of Optical Transport Networks for Grid Excess Load Handling*. Photonic Network Communications, 2006. Accepted for Publication.

[18] P. Thysebaert, F. De Turck, B. Dhoedt, and P. Demeester. *Using Divisible Load Theory to Dimension Optical Transport Networks for Computational Grids*. In Proceedings of OFC/NFOEC - on CD-ROM, 2005.

6 On-Line Grid Scheduling

6.1 Introduction

Chapter 1 has provided an overview of Grid [1] environments. These environments consist of a large number of heterogeneous resources, located at various Grid sites and connected through a wide area network infrastructure. Arguments - mostly based on the data-intensive nature of typical Grid applications - have been put forward in favor of increased adoption of interconnecting networks using optical technologies in this context [2, 3].

Ultimately, the massive processing power available in a Grid is exploited by co-allocating multiple resources (of different types e.g. computational resources, storage resources and network resources) to each job. The selection and allocation of these resources to jobs is done by a scheduler. As explained in chapter 2, these resource allocations are interdependent and therefore a scheduling model dealing with rigid and a priori specified per-job resource demands [4–7] does not capture the entire complexity of the Grid scheduling problem. In addition, it was shown that an off-line scheduling model can benefit from treating the workload as arbitrarily divisible [8–10].

Such a model was briefly touched upon in chapter 5 in the context of a combined off-line lambda Grid dimensioning and scheduling problem. In this chapter, we first re-iterate the problem of an off-line Grid scheduling model. Starting with a rigid project scheduling problem as described in chapter 2, we show how to extend this model to incorporate the notion of resource allocation interdependence

(specifically the interdependence between computational resource and network resource allocations). As in the previous chapter, we assume the network to be an optical circuit switched network meaning that our model contains wavelength continuity and granularity constraints.

Next, we invoke the notion of an arbitrarily divisible workload again to obtain a more scalable linear program representing the off-line Grid workload scheduling problem. As has been noted in chapter 5, the Grid dimensioning problem depends on the workload the Grid is to process and the way this workload is scheduled. As such, the off-line scheduling model presented in this chapter can be retrieved (when applied to the workload in question) in the combined lambda Grid dimensioning and excess load scheduling problem from the previous chapter.

While an off-line solution to the Grid workload scheduling problem is of use to the off-line dimensioning problem, workload is scheduled on an operational Grid by the scheduler(s) in an on-line fashion. In this chapter, several such on-line scheduling policies are studied. We specifically focus on steady state Grid workloads resembling (in size and resource requirements) the workload for which the Grid has been dimensioned - recall that from our discussion on the combined lambda Grid dimensioning and scheduling problem, the optimal off-line distribution of this workload follows from the solution to this problem.

The on-line scheduling policies mentioned here attempt to make use of this off-line solution when selecting and allocating resources to incoming jobs. In order to deal with scalability issues arising from the use of a fully centralized scheduling system in an operational Grid, we consider a two-level scheduling approach where resource selection and the interdependent allocation of multiple resources are dealt with separately. The algorithms we compare in this chapter specifically deal with the resource selection process. Our comparison uses various metrics and cost functions, and we investigate the added value of incorporating the off-line optimal workload distribution in the on-line scheduling process.

The resource models and lambda Grid interconnection topologies used in our evaluations are similar to the ones used in previous chapters.

6.2 Models

A complete description of an on-line Grid scheduling framework comprises 4 major components [11]:

- the application model
- the resource model
- the scheduling policy
- the performance model

The major properties of these components correspond to the Grid resource properties outlined in chapter 3, as we have used NSGrid for all simulations presented in this chapter, and are summarized below.

6.2.1 Application Model

In this chapter, we use the same application and job models as explained in section 3.3.8 and figures 3.6 through 3.8, as it clearly reflects the resource allocation interdependence problem.

When evaluating scheduling policies in this chapter, we limit ourselves to using jobs requiring a single input data set and emitting a single output data set.

6.2.2 Resource Model

The time-shared resource (computational, storage and data replica) models used here correspond to the resource models as described in sections 3.3.3 through 3.3.5.

As in the previous chapter, we will be dealing dealing with lambda Grids, which means the interconnecting network (between the participating Grid sites) comes in the shape of a circuit switched optical transport network. In order to transfer data between different sites, so-called *lightpaths* must be set up between these sites. The provisioning of an adequate number of lightpaths, the routing of lightpaths over the different network edges and optical cross connects and the exact wavelengths allocated to a lightpath are decided upon when dimensioning this network - the type of problem discussed in chapter 5. As this chapter deals with scheduling on an already deployed Grid infrastructure, it can be assumed that lightpaths have been setup and routed. It is assumed that a fixed-bandwidth window on these lightpaths can be allocated to the jobs to be scheduled (see section 3.3.2), much like the computational resource allocation is performed.

6.2.3 Scheduling Policies

The on-line scheduling policies detailed in section 6.4 operate as two-level hierarchical scheduling algorithms and schedule arriving jobs as soon as possible on a suitable set of resources (the criteria used to rank resources and assign them a level of suitability is what the algorithms differ in). The algorithms do not operate in *batch* mode (in which scheduling is performed at regular time instants and all unscheduled jobs are scheduled together), but rather perform resource selections as soon as a new job arrives.

Jobs cannot be pre-empted: once a job is started on its allocated resources, it runs on those resources until completion. This means that jobs do not migrate and cannot be interrupted (checkpointed) and rescheduled at a later time, even if such action would improve the resulting schedule.

The first level of the on-line scheduling framework concerns itself with resource *selection*. The metrics used to make this selection are described in the following section. Once a resource set has been selected, these resources are co-allocated to the job in order to minimize the job's *response time*. This resource *allocation* is performed by the local resource managers of the selected resources without involvement of the resource selection mechanism. The use of two distinct levels improves scalability of the scheduling infrastructure, as the top level does not need to concern itself with the exact resource time windows allocated to each job. This two-level approach makes the scheduling infrastructure more realistic; this is reflected in the fact that currently deployed Grid scheduling mechanisms need to interact with local schedulers and resource managers [12].

6.2.4 Performance Metrics

Both the off-line and on-line scheduling algorithms described distribute the Grid's *load* across all participating Grid resources. The *load* of a resource over some period of time is taken to be the amount of work performed by that resource during the observed time period (see figure 6.1) divided by the the resource's capacity, resulting in a number of "resource-seconds" - the minimal amount of time the resource needs to perform all of the work performed in the observed interval. Thus, the observed load is influenced by the time of observation, and the arrival and completion of jobs on the observed resource. Unless mentioned otherwise, in this chapter, the time interval over which a resource's load has been calculated upon arrival of a new job starts at that job's arrival time and ends at the current schedule's makespan. Due to the nature of the workloads studied here (with workload sizes approximating the Grid's capacity), distributing these workloads may lead to a near *load balancing* situation (which would be obtained exactly by minimizing the maximal load observed on any one resource in the Grid).

When observing the Grid in steady state, the load on a resource can be defined as the instantaneous amount of work being performed by the resource per time unit divided by the resource's capacity. For all resources, this yields a real number in $[0, 1]$ for its steady state load.

In both cases (i.e. with or without explicit time instants in the observations), these metrics allow us to compare (the load on) the various resources as these metrics are expressed in the same units (either time units or dimensionless numbers) regardless of the type of resource under investigation.

On-Line scheduling algorithms can be compared using the resource loads observed during the scheduling process and using the *response time* (the difference between completion and arrival times) experienced by the scheduled jobs making up the Grid's workload.

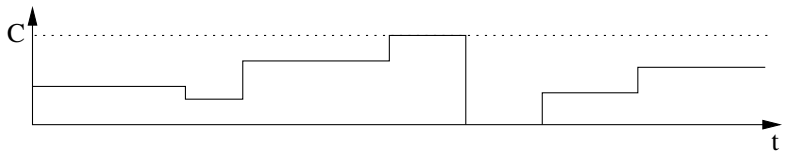


Figure 6.1: Work performed by Time-shared Resource with Capacity C over time

6.3 Off-Line Multi-Resource Scheduling

6.3.1 Off-Line Scheduling Formulation

Off-line scheduling problems in which each scheduling decision involves the allocation of multiple resources have been studied extensively in literature. Most important classes of multicomputer and multiprocessor scheduling problems can be described using linear programming as a special instance of a Resource Constrained Project Scheduling Problem (see chapter 2). This general case supports the description of workloads needing allocations with multiple resource types, where each resource requirements is known a priori.

While supporting multiple resource types is a fundamental requirement for every Grid scheduling model (as e.g. computational resources, storage resources and network links are all to be treated as first class resources in such an environment), Grid scheduling problems differ from project scheduling problems in a few ways.

First, resource allocations for a single Grid job requiring multiple resources may not be independent: Grid jobs which have been allocated only a small amount of CPU time probably cannot make full use of high bandwidth network connections to storage resources, as they simply do not have time to process data arriving at full rate.

In addition, the exact instantaneous resource demand of a Grid job is part of the scheduling process and not fully known in advance, in contrast to its fixed total resource demand. This is because the jobs can be scheduled on time-shared resources (the size of the share is determined in the scheduling process), and, because of the resource allocation interdependence mentioned before, this may affect all other resource allocations for that particular job as well.

In section 6.3.2, we model the Grid scheduling problem (with resource and application models as described in section 6.2) as an extended version of a Resource Constrained Project Scheduling Problem. The extensions have the explicit goal of taking into account the resource allocation interdependence and the dynamic nature of their sizes.

However, while this formulation does completely capture our model of the off-line Grid Scheduling problem in a linear program, we argue that this program quickly becomes intractable even for modest Grid sizes. Therefore, in sec-

tion 6.3.3, by making additional assumptions, we propose a more scalable version of this linear program by making use of divisible load theory.

6.3.2 RCPSP Model

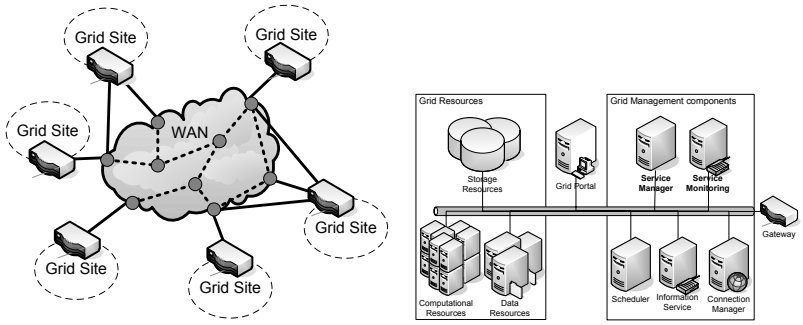
The off-line grid scheduling problem involving J jobs can be described as an extension of a Multi-Modal Resource Constrained Project Scheduling Problem (MMRCPSP) and thus modeled into an integer linear program, where each mode in which a job can be executed is the collection of resource allocations assigned to that job.

For this linear program, assume that each job j (which is submitted on its *home* computational resource h_j) is executed on a single computational resource and processes a single data stream, located at the site where this job was submitted. All output data generated by a job must be returned to its submission site, where it is presented to the user responsible for launching the job. The job j is launched at time r_j, its computational length is l_j, the amount of input data it processes is d_j^i bytes, and the amount of output data generated is d_j^o.

The Grid itself is modeled after figure 6.2 and adheres to the following properties:

- The Grid is made up of several *sites*; these sites are interconnected through an optical circuit switched (OCS) network, in which (virtual) lightpaths between sites have been established.
- Each site consists of a time-shared computational resource c with processing capacity C_c and a time-shared storage resource s with total storage capacity S_s, and is connected to the OCS network through a gateway.
- The intra-site networks are assumed to have sufficient capacity in order not to create bottlenecks.
- The number of (virtual) wavelength paths that have been setup between the sites hosting computational resources c and c' is $\lambda_{cc'}$. Each wavelength paths offers data rate B. The total number of wavelength paths that have been set up in the network is denoted L.

All of the above numbers are considered given and thus are input constants to the scheduling problem. The key concept in modeling the complete Grid scheduling problem (as opposed to a standard MMRCPSP) as an integer linear program is to associate a set of *dummy jobs* with each job j; the number of dummy jobs needed equals the number of resource types that need to be allocated by this job. For instance, focusing on data streaming jobs, 3 dummy jobs are needed for each real job, all of which are to execute simultaneously: one for the input data transfer, one for the output data transfer and one for the processing (in contrast, for



(a) Grid Model (b) Individual Grid Site Model

Figure 6.2: Grid and Grid Site Conceptual Models

data staging jobs, we would need the same three dummy jobs but with precedence relations between them).

Directly related to the dummy jobs are the decision variables needed in the problem: each dummy job needs to be scheduled on an available resource of the appropriate type (identified by the type of dummy job).

Assuming a discrete time scale, with increasing time instants of interest labeled $0, 1, \ldots, t, \ldots, T$ (these time instants need not be equidistant but can, for scalability reasons, as well be (geometrically) increasing in distance - see [6] for instance), this leads us to introduce binary decision variables $r^i_{jRbe}, r^o_{jRbe}, r^p_{jRbe}$ which equals one if and only if the dummy job of type i,o or p, respectively, for job j is executed on resource R starting no earlier than time instant b and ending no later than time instant e. The different resource types i, o, p point to network links for input data transfer, network links for output data transfer and computational resources used for the actual job processing. As we are dealing with discrete time instants, the interval $[t_b, t_e[$ is to be interpreted as being the smallest interval containing the affected job's life cycle in its entirety, rather than representing accurate values for the job's start and completion times.

If we want to obtain a load balancing schedule (we will not consider alternative factors such as resource usage cost), the following objective function may be used as a minimization criterium:

$$Objective = MaxLoad + P * Makespan \tag{6.1}$$

In this expression, "Makespan" refers to the resulting schedule's makespan, "MaxLoad" is the maximal load (over all resources) assigned to a resource and "P" is a penalty factor which, when taken large enough, enforces the selection of a load balancing schedule among all feasible schedules featuring minimal makespan by eliminating unnecessary idle time. These auxiliary quantities can be obtained from the

constraints

$$\forall j.Makespan \geq \sum_{c=1}^{C}\sum_{b=0}^{T-1}\sum_{e=b+1}^{T} t_e r^p_{jcbe} \tag{6.2}$$

$$\forall c.MaxLoad \geq \sum_{j=1}^{J}\sum_{b=0}^{T-1}\sum_{e=b+1}^{T} \frac{r^p_{jcbe} l_j}{C_c} \tag{6.3}$$

$$\forall \lambda.MaxLoad \geq \frac{\sum_{j=1}^{J}\sum_{b=0}^{T-1}\sum_{e=b+1}^{T} d^i_j r^i_{j\lambda be} + d^o_j r^o_{j\lambda be}}{B} \tag{6.4}$$

Capacity constraints on the various resources are given by

$$\forall c.\forall t.\sum_{j=1}^{J}\sum_{b=0}^{t}\sum_{e=t+1}^{T} \frac{l_j r^p_{jcbe}}{t_e - t_b} \leq C_c \tag{6.5}$$

$$\forall \lambda.\forall t.\sum_{j=1}^{J}\sum_{b=0}^{t}\sum_{e=t+1}^{T} \frac{d^i_j r^i_{j\lambda be} + d^o_j r^o_{j\lambda be}}{t_e - t_b} \leq B \tag{6.6}$$

$$\forall s.\sum_{j:s\in h_j} d^o_j \leq S_s \tag{6.7}$$

Since a job j only arrives at time r_j, it cannot be started before that time:

$$\forall j.\sum_{c=1}^{C}\sum_{b:t_b<r_j}\sum_{e=b+1}^{T} r^p_{jcbe} = 0 \tag{6.8}$$

A unique schedule is obtained if

$$\forall j.\sum_{c=1}^{C}\sum_{b=0}^{T-1}\sum_{e=b+1}^{T} r^p_{jcbe} = 1 \tag{6.9}$$

$$\forall j.\forall c.\forall b.\forall e.r^p_{jcbe} = \sum_{\lambda:h_j\to c} r^i_{j\lambda be} \tag{6.10}$$

$$\forall j.\forall c.\forall b.\forall e.r^p_{jcbe} = \sum_{\lambda:c\to h_j} r^o_{j\lambda be} \tag{6.11}$$

The previous linear program's computational complexity is summarized in table 6.3.2.

# Variables	# Constraints
$\frac{(C+2L)JT(T+1)}{2}$	$3J + C + (T+1)(C + L + JCT)$

Table 6.1: Off-Line Grid Scheduling as an extension of MMRCPSP: linear program size

6.3.3 DLT Model

Two obvious causes for the rapid intractability (for increasing Grid sizes) of the previous linear program are the number of jobs involved and the number of discrete time instants of interest, as the program's complexity increases proportionally to these parameters.

The concept of divisible load [8, 9] explicitly deals with these hurdles in the context of off-line scheduling problems by making the following assumptions:

- the Grid system, on which workload is to be scheduled, is analyzed in *steady state*
- the workload to be scheduled is assumed to be arbitrarily divisible

The first assumption (restricting the scheduling problem's analysis to steady state Grids) eliminates the need to investigate discrete time instants. Instead, only the continuous quantities representing the amount of workload *per unit of time* being moved around are of importance.

Assuming that the workload is arbitrarily divisible means that only the aggregate workload arriving over some time window is of importance and not the individual jobs it is made up of - we will use α^c to denote this aggregate workload arriving per time unit at computational resource c. This ensures that the scheduling problem can be described by real-valued variables $\alpha_{c'}^{c} \geq 0$, denoting the amount of computational workload arriving per time unit at computational resource c (in that workload's home site) which is ultimately processed by remote computational resource c', and eliminates the need for per-job decision variables.

Again, we will not take into account resource costs, but rather attempt to find a load balancing schedule. This is equivalent to minimizing the maximal load found on any one resource, and this maximal resource load (relative to each resource's capacity) can be obtained from constraints of the form

$$\forall c. Load \geq \frac{\sum_{c'} \alpha_c^{c'}}{C_c} \tag{6.12}$$

$$\forall c. \forall c' \neq c. Load \geq \frac{\alpha_{c'}^{c} d_c^i + \alpha_c^{c'} d_{c'}^o}{B\lambda_{cc'}} \tag{6.13}$$

The resource capacity constraints now become

$$\forall c. \sum_{c'} \alpha_c^{c'} \leq C_c \tag{6.14}$$

$$\forall c. \forall c' \neq c. \alpha_{c'}^{c} d_c^i + \alpha_c^{c'} d_{c'}^o \leq B\lambda_{cc'} \tag{6.15}$$

In the last equation, d_c^i and d_c^o represent the average input and output data set sizes for jobs arriving at Computational Resource c, divided by the average computational length of these jobs.

A meaningful schedule is obtained if

$$\forall c. \sum_{c'} \alpha_{c'}^{c} = \alpha^{c} \tag{6.16}$$

with α^c denoting the total workload arriving at computational resource c per time unit.

If only the excess load is modeled in the problem as α^c, all that needs to change is that we should avoid the case where $c' = c$ in the previous constraints. If a single source excess load scenario is studied, this will be reflected in the fact that only one of the input constants α^c differs from zero.

Modeling the off-line Grid scheduling problem using the divisible load approach yields a linear program with lower computational complexity, as shown in table 6.2.

# Variables	# Constraints
C^2	$C(2C+1)$

Table 6.2: Off-Line Grid Scheduling using Divisible Load: linear program size

6.4 Two-Level On-Line Scheduling Algorithms

6.4.1 On-Line Scheduling Framework

Off-Line scheduling models like the ones presented in section 6.3 commonly appear in the scope of a Grid dimensioning problem [13]. In such an off-line dimensioning problem, each resource's capacity is a decision variable rather than an input constant. The exact computational capacity and (in a lambda Grid) number of lightpaths to be installed between sites depends on the envisioned job schedule, as these scheduling decisions directly influence the necessary (remote) processing power and network bandwidth.

Once a Grid has been deployed, arriving jobs will of course be scheduled on this infrastructure following an *on-line* scheduling policy. While on-line scheduling algorithms and heuristics have been studied extensively in literature [4–6], most concentrate on problems with a single resource type (i.e. CPU time).

Since multiple resource types are a key element in every Grid, scheduling heuristics taking into account one resource at a time are not the most appropriate for the Grid scheduling problem. Moreover, many of these approaches lack a sound mathematical foundation as resource usage and cost for different resource types are incompatible and involve different dimensions.

To tackle this problem, a *unified* approach to modeling resource assignment cost was proposed by Keren et al. in [14]. They describe a on-line scheduling

framework which allows for the inclusion of different resource types by using dimensionless quantities, in particular the work assigned to a resource divided by that resource's capacity. In addition, an optimization goal based on economic principles and marginal cost analysis is proposed as an improvement of a greedy list scheduling algorithm.

Because of the sheer size of typical Grids, it is unrealistic to envision a fully centralized scheduling system responsible for managing and co-allocating resources to every job. Instead, as explained in section 6.2.3, it is more reasonable to assume a hierarchical two-level scheduling model where a top level scheduling system makes use of local resource schedulers.

The on-line load-balancing algorithms examined in this section are situated at this top level scheduling system. First, we will show how the unified on-line framework can be used within the Grid scheduling model described above. In particular, we describe how the greedy algorithm and the marginal cost based algorithm have been implemented for our simulations. These algorithms are based purely on current resource state: they do not rely on information concerning resource usage history or future job properties.

The last class of algorithms we present not only uses resource state information, but also employs results obtained by solving the off-line Grid scheduling problem developed in section 6.3.3. These results represent the optimal steady state Grid workload distribution for the workload for which the scheduling problem was solved (in the context of a dimensioning problem, this workload would commonly represent the most stressing load the Grid needs to be able to handle). This optimal steady state off-line workload distribution is used in the algorithm as a *target* load, and the algorithm's optimization goal is to mimic this target load in an on-line fashion.

6.4.2 Greedy Scheduling Algorithm

The *greedy* on-line scheduling algorithm schedules jobs as soon as they arrive; suitable resources are selected by attempting to minimize the resulting schedule's (i.e. the schedule consisting of the newly arrived job and all jobs which have already been allocated) maximal load on any single resource.

Formally, the greedy algorithm attempts to obtain a load-balancing schedule by selecting the resource set $\mathcal{R}_j$ for a newly arriving job j such that the quantity

$$\max_{r \in \mathcal{R}} l_r^{\mathcal{R}_j} \tag{6.17}$$

is minimized, where $l_r^{\mathcal{R}_j}$ denotes the load on resource r given that job j is scheduled on the resources contained in the set $\mathcal{R}_j$. If this maximal resulting resource load is equal for two different resource assignments, the load of the resource having the second highest load is compared and so on. This approach can be used

(and is implemented as such) as a comparison operator for the resource load *vectors* obtained with each resource assignment.

The load for a single resource is calculated as specified in section 6.2.4, where the observed time interval is taken to start at the new job's arrival time and ends at the schedule's makespan.

6.4.3 Opportunity Cost based Scheduling Algorithm

The opportunity cost algorithm also schedules jobs as soon as they arrive, but uses an objective function based on marginal cost analysis [14] to select a suitable resource set. The previous algorithm, greedy, essentially tracks the resulting increase in resource load induced by every possible scheduling decision.

In the opportunity cost algorithm, the quantity to be minimized (the marginal cost of the resource assignment) is given by

$$\sum_{r \in \mathcal{R}} \left(a^{l_r^{\mathcal{R}_j}} - a^{l_r} \right) \tag{6.18}$$

where $l_r^{\mathcal{R}_j}$ again denotes the load on resource r after job j has been scheduled on the resources contained in set $\mathcal{R}_j$ and l_r now denotes the load on resource r *before* job j had been scheduled on the same set of resources. The parameter a used in this formula is a constant > 1.

Note that this cost function is not only sensitive to the increase in resource load, but also in the size of the increase relative to the current load allocated to the resource.

6.4.4 DLT based Scheduling Algorithms

The DLT based on-line scheduling algorithms get additional input (the steady state *target* load for each resource) from the solution to the off-line scheduling problem (e.g. performed when the Grid is being dimensioned) as described in section 6.3.3.

The cost function associated with a particular resource assignment for job j can take on the form

$$\max_{r \in \mathcal{R}} d_r^{\mathcal{R}_j} \tag{6.19}$$

or, using the marginal cost approach,

$$\sum_{r \in \mathcal{R}} \left(a^{d_r^{\mathcal{R}_j}} - a^{d_r} \right) \tag{6.20}$$

In the former case, as with the Greedy algorithm, the quantity of interest is actually a *vector* rather than a scalar.

In the above expressions, d_r is the *deviation* from the target load observed on resource r before job j has been scheduled on resource set $\mathcal{R}_j$, and $d_r^{\mathcal{R}_j}$ is the deviation from resource r's target load observed on it in the resulting schedule.

Given the target steady state load t_r for resource r and currently observed load l_r, we have used the following alternatives to define deviation d_r:

$$d_r = \frac{l_r}{t_r} \tag{6.21}$$

$$d_r = |t_r - l_r| \tag{6.22}$$

$$d_r = \max(0, l_r - t_r) \tag{6.23}$$

Again, the proposed definitions and equations are mathematically sound in the sense that only dimensionless numbers are used when calculating costs involving different resource types.

Note that, given the previous definitions of the deviation concept, a resource's target load deviation does not necessarily increase when assigning extra load to this resource - in contrast to the total workload assigned to the resource. Therefore, for the rest of this chapter, we will not pursue the use of the marginal cost approach when resource target load deviation is the metric of choice as this approach requires an increasing cost function.

As with the previously presented algorithms, the scheduling algorithm elects the resource set yielding minimal cost.

It is worth noting that, although the divisible load based on-line algorithms use more information when compared to the other algorithms, this does not result in increased computational complexity as the required information has been calculated and made available from the off-line combined scheduling and dimensioning problem.

6.5 Evaluation: Setup

The main experiments performed and described in this chapter pertain to the evaluation and comparison of the hierarchical two-level on-line scheduling algorithms presented in sections 6.4.2-6.4.4 and their comparison to the fully centralized single-level algorithm which schedules incoming jobs on the resource set yielding minimal completion time in a greedy fashion. Due to the nature of the divisible load based algorithms, the experiments consist of two phases:

- Solving an off-line steady state scheduling problem (in the context of a Grid dimensioning problem)
- Simulating the on-line scheduling of the appropriate workload (approximated in the first phase) on the Grid used in the first phase

The last step - the workload scheduling - consists of the resource selection and allocation phases as indicated in section 6.2.3. Our experiments have been performed for a wide range of parameter variations: for our simulations, we have varied the Grid interconnection network connectivity, the stochastic workload used during the dimensioning and scheduling phases and - for the divisible load based algorithms - the definition and implementation of the divisible load cost function and deviation metric.

6.5.1 Simulated Topologies

Our simulations have been performed for different Grid interconnection (i.e. Optical Transport Network) topologies, inspired by the sample European network shown in figure 6.3 and introduced in section 5.4.2.

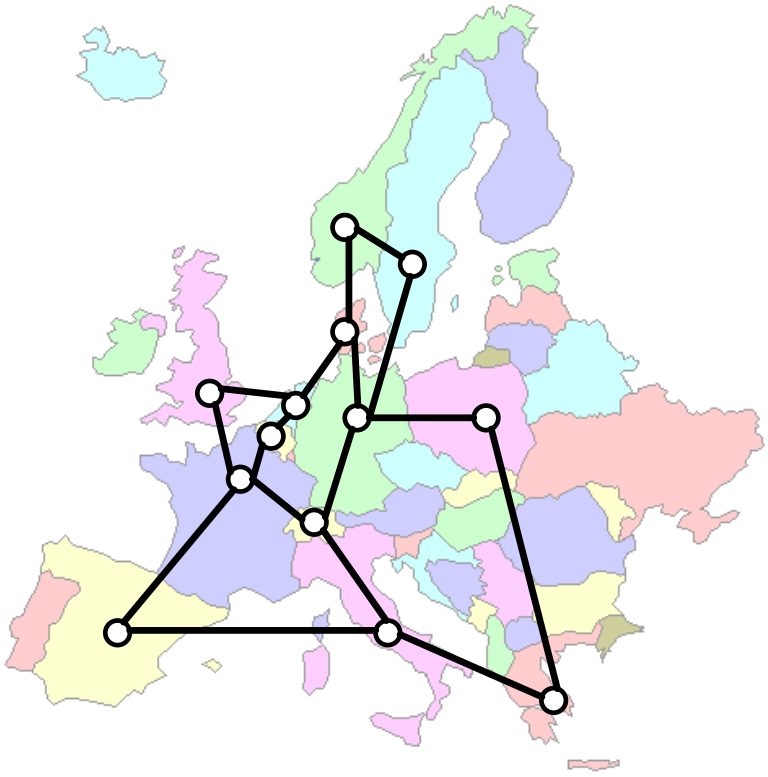


Figure 6.3: 13-Node European Network

We used the sets of random networks as described in section 5.4.7. These random networks (all having 13 nodes) were generated by constructing a connected

set of 13 nodes through repeated addition of node-link pairs, and then added in extra links following a probability p in $[0, 1]$. This parameter was varied to generate networks with different average connectivity values. All Grid topologies have been dimensioned so that excess load generated at one site can be handled by the aggregation of all remote sites. We have chosen 13 computational resources with geometrically increasing processing capacity, with a reference capacity of 3 units of work per time unit and a capacity increase of 5% between successive computational resources.

6.5.2 Simulated Scenarios

The scheduling scenarios studied are those found in two-tier Grids. In particular, we study scenarios where excess workload arrives at one site (the top tier) which must then be distributed to the other, remote sites (the second tier). In case of uniform workload distribution, these scenarios reduce to the scenarios detailed in section 5.2.3. The scenarios used in this chapter differ from the scenarios described in section 5.2.3 in that uniform workload distribution is not enforced. Rather, we look for the optimal excess workload distribution over the remote sites.

In the dimensioning phase of our experiments, each Grid topology (as described in the previous section) has been dimensioned in order to support all possible two-tier excess load scenarios as described above. This means that the off-line combined Grid dimensioning and scheduling problem solved describes - in our case - thirteen workload scenarios. Again, the computational resources have been dimensioned to handle load up to the 60% percentile of the distribution describing the arriving workload at that resource's site. Excess load is then created by having one site generate more workload than can be handled by its own computational resource. The simulation results presented below were obtained by scheduling the same excess workload in these thirteen scenarios on the resulting Grid.

In order to measure the relevant metrics in steady state, transient effects occurring in the start and end phases of each simulation (corresponding to workload build up and workload draining mechanics in the Grid) have been eliminated from our calculations.

6.5.3 Simulated Workload

The workload used in the scheduling simulations is chosen in such a way that the excess load arriving per time unit at the single source site in each scenario equals 90% of the Grid's residual computational capacity. To process this amount of excess load, co-operation of all remote Grid sites is required. An on-off distribution was used to model the intermittent arrival of large and small jobs. The distributions from which excess job parameters were drawn are shown in table 6.3. We have used a standard workload and a workload featuring the same average values,

yet bigger variance for its constituent parameters. This latter workload is mainly used in section 6.6.4. Distributions are taken to be uniform. For use in our DLT based dimensioning and scheduling model, the parameters shown in table 6.3 yield the same average amounts of arriving computational load per time unit as well as the same average amount of data transferred per unit of processing. Our on-line workload consisted of 1000 jobs. To eliminate transient scheduling and queue draining effects, performance metrics such as average job response time have been measured without taking into account the first and last 100 jobs.

Workload	On-period (Jobs)	On-Joblength	On-Data(MB)	Off-Joblength	Off-Data(MB)
Standard	3/10	40-50	5000-9000	10-17	5000-9000
Inc. $\frac{\sigma}{\mu}$	1/10	100-150	5000-9000	10-13.2	5000-9000

Table 6.3: Excess Workload Characteristics

In our scenarios, the average job interarrival time was taken to be $0.5s$. Each job reads its input from and returns it output to its submission site. The Grid's network has been dimensioned (see section 6.6) to support the average expected network load resulting from this.

6.6 Evaluation: Results and Discussion

6.6.1 Grid Interconnection Network Dimensioning

For different average job interarrival times, the resulting Grid network dimensioning cost has been shown in figure 6.4 for two different interconnection topology average connectivities. For larger interarrival times, less workload arrives in the Grid and thus less network traffic is generated, resulting in smaller (i.e. cheaper) network capacity to be installed. At the same time, networks with lower connectivity yield longer paths between node pairs, explaining the higher network cost obtained for the networks with $p = 0.1$. As stated in section 6.5.3, for our experiments excess job load is generated with an average job interarrival time of $0.5s$.

6.6.2 Job Response Time

In figures 6.5 and 6.6 the resulting schedule's average job response time in steady state has been plotted for two sets of random networks (ten in each set), having connectivity parameter p set to 0.1 and 0.9, respectively. Along the x-axis is the average job interarrival time; as discussed in section 6.5.3, the Grids in this chapter have been dimensioned in such a way that excess workload as described in table 6.3 and arriving with an average interarrival time of $0.5s$ equals 90% of the Grid's residual computational capacity. We have scheduled the excess load as described in section 6.5.3 in 13 different scenarios (each scenario corresponding to

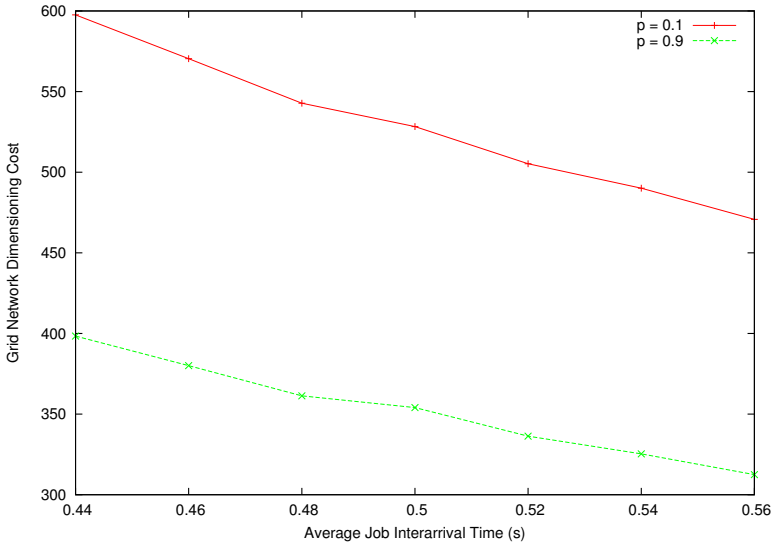


Figure 6.4: Resulting Grid Dimensioning Cost

a different excess load source node) in all networks and values presented are the averages over these 13 scenarios. As expected, the fully centralized single-level algorithm (performing an exhaustive search) does the best job in minimizing the average job response time, but is closely followed by the hierarchical algorithms.

The terms "DLT Diff", "DLT Frac" and "DLT Overload" refer to the use of deviation definitions 6.22, 6.21 and 6.23, respectively. From the figures, it is clear that no single two-level algorithm outperforms the others by a significant margin for the workloads of interest (around the average job interarrival time of $0.5s$). For the workload on which the Grid network dimensioning was performed (around an interarrival time of $0.5s$), the greedy, opportunity cost, DLT Diff, DLT Frac and DLT Overload scheduling algorithms show the same performance. For interarrival times $\ll 0.5$, the workload arriving approaches and ultimately exceeds the Grid's capacity. Furthermore, in this area it becomes clear that the arriving workload is in reality not arbitrarily divisible, as response times rise before the workload equals the total Grid capacity, and the Grid no longer remains in steady state (so in this area, average response times will continue to rise with increasing simulation time). For $l_r \ll t_r$, a case which frequently occurs when workload is low (i.e. interarrival times are high), the DLT-based algorithms behave worse than their non DLT-based counterparts as for these workloads the target workloads t_r no longer provide a realistic goal. This is especially pronounced when deviation definition 6.22 is used, as using this definition will (erroneously) assign high deviation values for those resources where $l_r \ll t_r$ impeding correct resource selection.

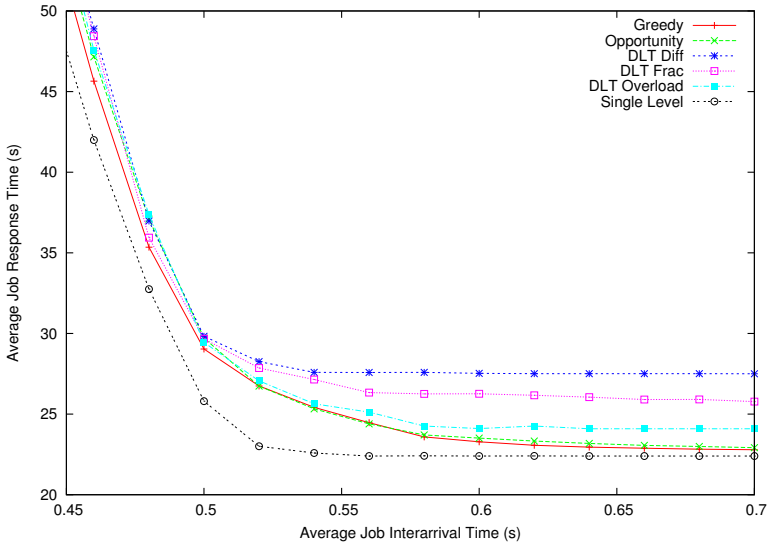


Figure 6.5: Job Response Times: Single-Level vs. Hierarchical Algorithms, $p = 0.1$

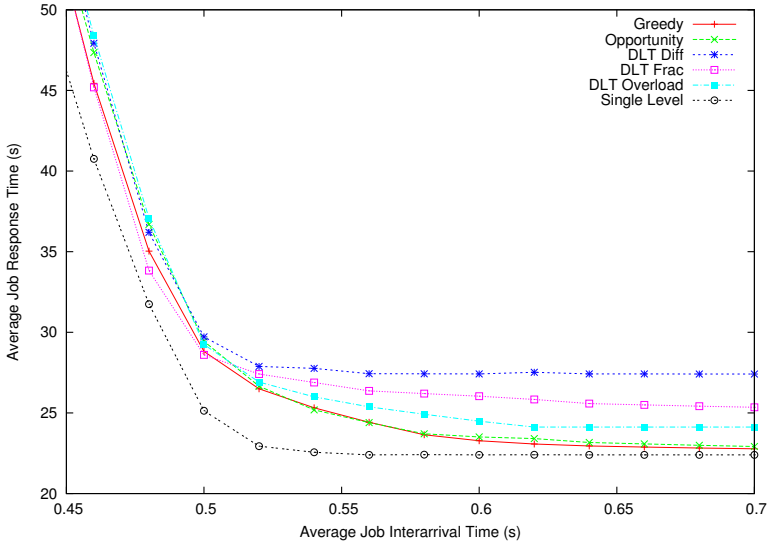


Figure 6.6: Job Response Times: Single-Level vs. Hierarchical Algorithms, $p = 0.9$

As can be expected because the interconnecting topology is only of importance during the dimensioning phase (during the scheduling phase, only lightpaths are of importance, not their routing over the underlying optical transport network), a

similar behavior is observed in both cases ($p = 0.1$ and $p = 0.9$) presented.

6.6.3 Resource Target Load Difference

Further comparing the different algorithms, at each job's arrival we have constructed a vector containing, for each resource, the *deviation* (as defined in equation 6.22) of the resource's current load from that resource's steady state target load. The average norm of this vector in steady state has been plotted in figures 6.7 and 6.8 for the different algorithms. Note that, according to definition 6.22, both underloaded and overloaded resources contribute to the deviation, which explains why the metric is always strictly positive.

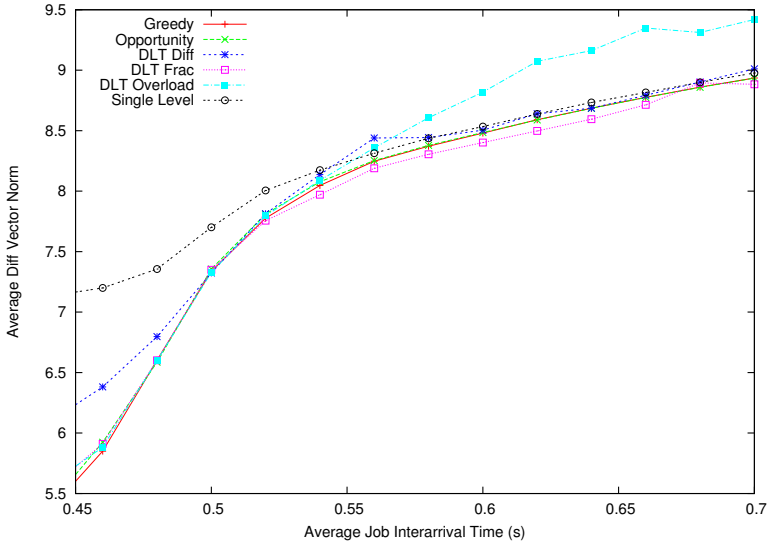


Figure 6.7: Diff Vector Norm at Job Schedule Time: Single-Level vs. Hierarchical Algorithms, $p = 0.1$

Again, the two-level scheduling algorithms exhibit the same values for this metric around the main point of interest, which is the workload obtained for an average interarrival time of $0.5s$. The single-level algorithm, which attempts to minimize each job's response time will make more use of faster computational resources than prescribed by the off-line target loads, and vice versa for the slower computational resources. This explains why it generates a schedule deviating more from the off-line calculated resource target loads than the schedules calculated with the other algorithms. Around $0.5s$, the DLT-based algorithms follow the prescribed target load more closely, which is important for steady state operation, as the steady state is guaranteed when the prescribed target load is matched exactly.

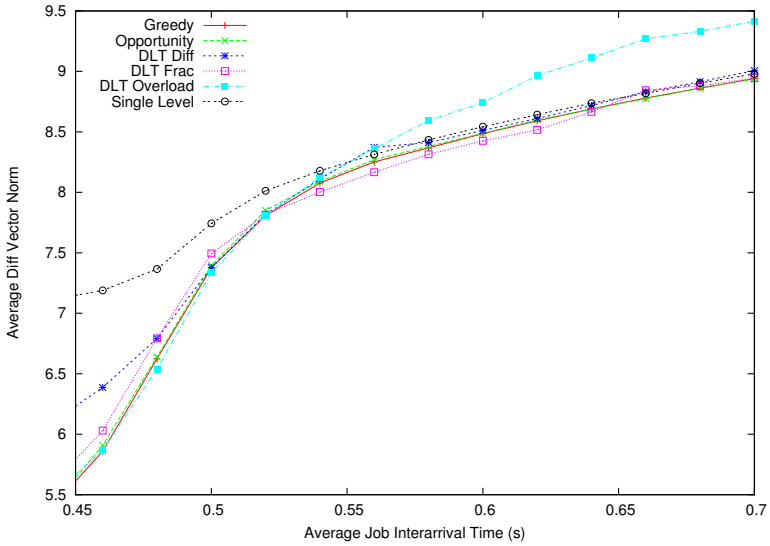


Figure 6.8: Diff Vector Norm at Job Schedule Time: Single-Level vs. Hierarchical Algorithms, $p = 0.9$

For low workloads (i.e. interarrival times $\gg 0.5$), the biggest deviation is obtained for the "DLT Overload" algorithm which doesn't care about unused resources who do contribute significantly to equation 6.22.

6.6.4 Job Length Distribution

The above results were obtained for a stochastic job generating process. In this section, we repeat the simulations performed earlier for a different job generating process. The resulting job set features the exact same *average* length, data sizes and interarrival time, but the distributions used now have a higher standard deviation than before (see section 6.5.3). As the average values remain constant, however, there is no need to re-dimension the Grid as the divisible load parameters remain the same. Again, our workload consisted of 1000 of such jobs of which the first and last 100 have not contributed to our measurements for reasons explained in section 6.5.3. We used the same sets of 10 random networks corresponding to values for p equal to 0.1 and 0.9. The results for these new simulations are shown in figures 6.9, 6.10, 6.11 and 6.12. As our arriving computational load exhibits more burstiness than the load used in section 6.6.2, resulting average response times rise more quickly for higher workloads and it is clearly visible how the operational Grid now leaves it steady state faster for increasing load. In the workload area where the Grid can be maintained in steady state, our algorithms exhibit the same performance relative to one another. Again, the single-level scheduling algorithm

provides the smallest average job turnaround times, but does not necessarily mimic the calculated target load distribution in doing so. For small workloads (when the average interarrival time $\gg 0.5$, resulting in $l_r \ll t_r$) the algorithms yield average response times comparable to those shown in figures 6.5 and 6.6. This follows from the fact that, although the job length used in this section has a higher standard deviation than the standard workload used throughout section 6.6.2, the *average* job length (and thus, for equal average interarrival times, the average amount of computational load arriving per unit of time) are the same. In figures 6.11 and 6.12, the resource target load metrics for the different algorithms match best around an average interarrival time of 0.6, which roughly corresponds to the minimal interarrival time for which the Grid remains in steady state.

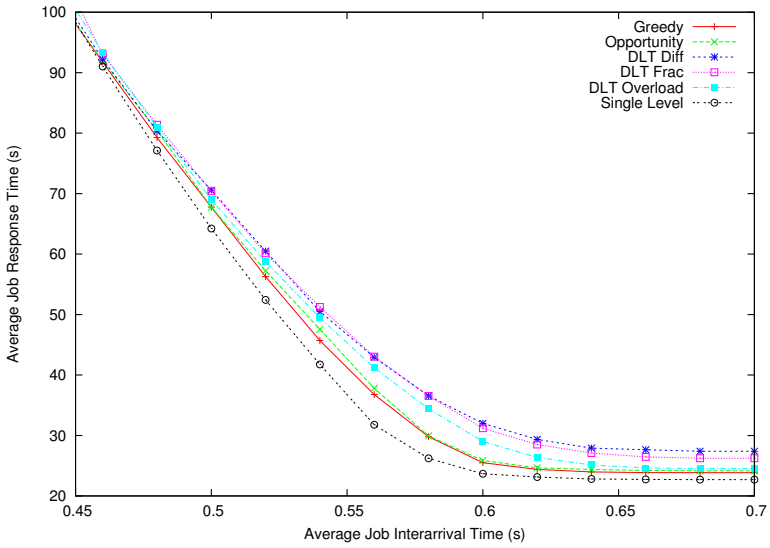


Figure 6.9: Resulting Job Response Times: Increased Job Variability, $p = 0.1$

6.7 Conclusions

In this chapter, we have first shown how to extend classical resource project scheduling problems as to incorporate the notion of interdependent resource allocations, specific to Grid workload scheduling problems. Next, we have proposed to use the concepts of an arbitrarily divisible workload to reduce the off-line scheduling problem's complexity.

The core idea presented in this chapter is to make use of the off-line optimal workload distribution when scheduling workload in an operational Grid, maintaining a stable steady state. To this end, we have concentrated on two-level hi-

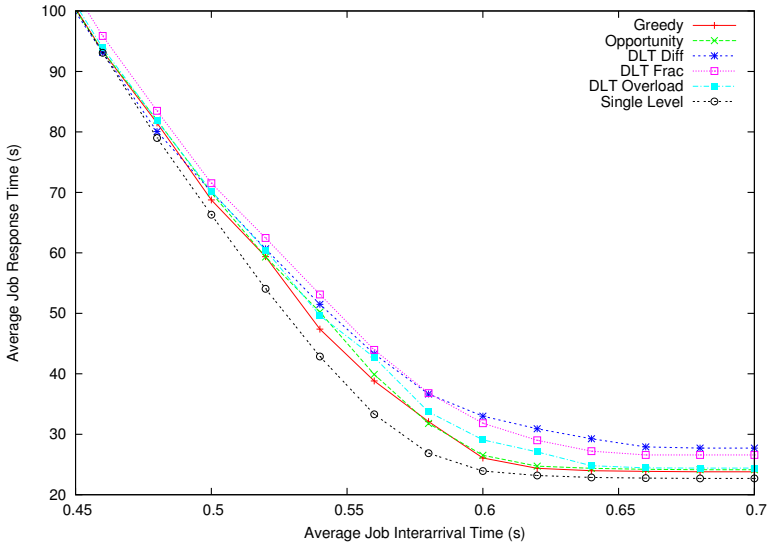


Figure 6.10: Resulting Job Response Times: Increased Job Variability, $p = 0.9$

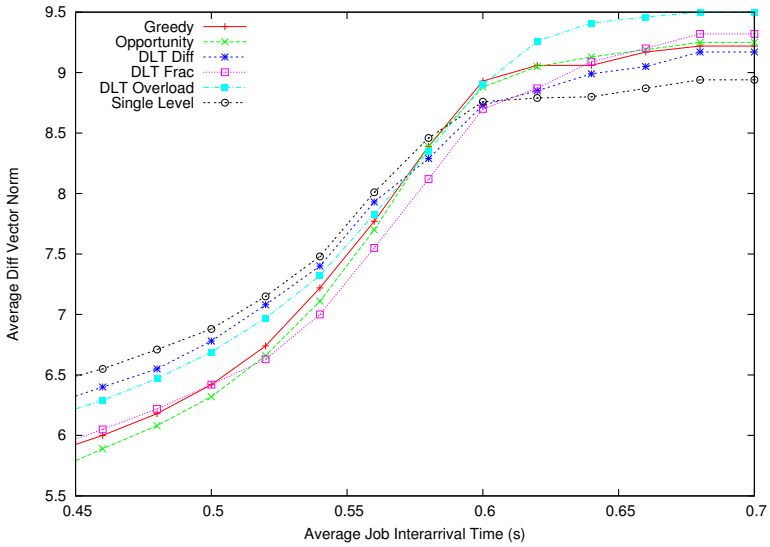


Figure 6.11: Diff Vector Norm at Job Schedule Time: Increased Job Variability, $p = 0.1$

erarchical on-line scheduling policies, separating resource selection and resource allocation, as a fully centralized approach is likely to lack in scalability and is therefore not suited to be deployed in a Grid environment.

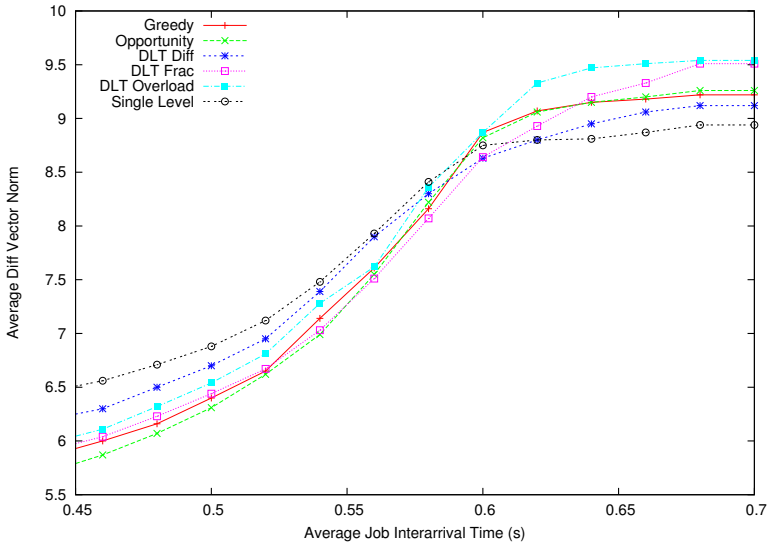


Figure 6.12: Diff Vector Norm at Job Schedule Time: Increased Job Variability, $p = 0.9$

The scenarios studied correspond to that of a lambda Grid featuring a single excess load source where the workload closely resembles the worst-case workload used during the Grid dimensioning phase. When scheduling workload in these scenarios, the off-line calculated workload distribution has been used as resource load target in the on-line resource selection algorithms we presented.

For different Grid interconnection topologies, job generating processes and load metrics, we compared these algorithms to other (such as greedy and opportunity cost based) hierarchical scheduling algorithms (which do not explicitly take into account the off-line calculated workload distributions) as well as a single-level scheduling policy, and we have assessed the gain obtained by using the off-line calculated workload distribution as target load.

We found that, in the studied scenarios, using the off-line calculated workload distributions in an on-line algorithm in the way we described allows the Grid to operate close to the design point. In all of our experiments, a standard greedy resource selection policy is able to provide a good average job response time (and thus, keep the Grid operation in steady state) which does not differ much from the job response times obtained by using a single-level algorithm. However, due to scalability and computational constraints, this single-level algorithm is unlikely to be implemented in an operational Grid. Similar observations can be made when comparing the algorithms using deviation from the off-line calculated steady state workload distribution as a metric. Our DLT-based algorithms, however, incor-

porate the additional workload information in the scheduling process without increasing its computational complexity, and offer reasonable performance (when measured using job response times and target load deviation) when compared to the optimum delivered by the single-level scheduling algorithm.

References

[1] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure 2nd Edition*. Morgan Kaufmann, 2003.

[2] T. DeFanti, C. de Laat, J. Mambretti, K. Neggers, and B. Arnaud. *TransLight: A Global-Scale LambdaGrid for e-Science*. Communications of the ACM, 46:34–41, 2003.

[3] L. Smarr, A. Chien, T. DeFanti, J. Leigh, and P. Papadopoulos. *The OptIPuter*. Communications of the ACM, 46:58–67, 2003.

[4] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein. *Scheduling to minimize Average Completion Time: Off-line and On-line Approximation Algorithms*. Mathematics of Operations Research, 22(3):513–544, 1997.

[5] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, and P. Wong. *Theory and Practice in Parallel Job Scheduling*, pages 1–34. Springer Verlag, 1997.

[6] J. Sgall. *On-Line Scheduling - A Survey*. Lecture Notes in Computer Science, 1442:196–231, 1998.

[7] R. Kolisch and R. Padman. *An Integrated Survey of Project Scheduling*. OMEGA International Journal of Management Science, 29(3):249–272, 2001.

[8] D. Yu and T.G. Robertazzi. *Divisible Load Scheduling for Grid Computing*. In Proceedings of the IASTED 2003 International Conference on Parallel and Distributed Computing and Systems (PDCS), 2003.

[9] J.T. Hung, H.J. Kim, and T.G. Robertazzi. *Scalable Scheduling in Parallel Processors*. In Proceedings of the 36th Annual Conference on Information Sciences and Systems (CISS'02), pages 20–22, 2002.

[10] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. *A realistic network/application model for scheduling divisible loads on large-scale platforms*. Rapport de recherche de l'INRIA-Rhone-Alpes (RR-5197), 2004.

[11] Y. Zhu. *A Survey on Grid Scheduling Systems*. PhD Qualifying Exam Submission, Computer Science Department of Hong Kong University of Science and Technology, 2003.

[12] K. Ranganathan and I. Foster. *Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids*. Journal of Grid Computing, 1:53–62, 2003.

[13] P. Thysebaert, M. De Leenheer, B. Volckaert, F. De Turck, B. Dhoedt, and P. Demeester. *Scalable dimensioning of Optical Transport Networks for Grid Excess Load Handling*. Photonic Network Communications, 2006. Accepted for Publication.

[14] A. Keren and A. Barak. *Opportunity Cost Algorithms for Reduction of I/O and Interprocess Communication Overhead in a Computing Cluster*. IEEE Transactions on Parallel and Distributed Systems, 14(1):39–50, 2003.

7 Conclusions

In this research work the use of optical circuit switched interconnection networks in Grids (so-called lambda Grids) has been investigated. In particular, the problems of dimensioning these Grids as well as suitable algorithms for scheduling and distributing workload on these Grids have been studied in detail.

In order to perform this study, we have developed the following key concepts:

- a network aware Grid simulator called NSGrid providing accurate Grid resource and network models
- a scalable linear model for the combined off-line dimensioning and scheduling problem in these Grids
- a set of on-line workload scheduling algorithms based upon the solution to the previous off-line problem

When the lack of a Grid simulator providing accurate models for Grid resources, middleware components, network elements and transport protocols was discovered, work was started on the development of NSGrid, aimed to function as a performant Grid simulation environment providing all of these models. As accurate models for network elements and transport protocols are readily available in the well-known ns-2 network simulator, ns-2 was used as the base platform to jump start NSGrid's development.

NSGrid extends ns-2 by providing detailed models for computational, storage and data replica resources. In addition, it features behavioral models for the

most important Grid middleware components: Grid schedulers, network connection managers, information resources and monitoring components. We discussed in great detail the job model (supporting both streaming and pre-staged data models) assumed throughout our Grid simulations; to complete the Grid simulation environment, a dedicated component modeling a Grid end user submitting such jobs was also added to NSGrid.

We demonstrated two cases in which NSGrid was used to obtain quantitative results on a Grid operational scenario. The first case showed the importance of using network aware scheduling algorithms in Grid environments where jobs typically process and transfer large data sets. Whether jobs continuously access data streams or rather completely separate data transfer and execution, job turnaround times can be significantly reduced by using network aware scheduling. In addition, if different job classes with significantly different resource requirements are present, upfront bandwidth allocation to each class can improve the job response time by offering guaranteed progress to each of the job classes.

In the second case, we measured - through NSGrid simulation - the impact of different resource partitioning strategies whose job it is to instantiate several "Virtual Private Grids" based upon characteristics of different job service classes. We have shown how exclusively reserving resources for a single job class (by upfront Grid partitioning) results in better job response times (up to 22.6% when network aware scheduling is employed).

While NSGrid can simulate an operational Grid, it is of course limited to simulating on-line workload scheduling algorithms. In order to be able to compare these algorithms to optimal, off-line behavior we developed such an off-line model using linear programming. The off-line scheduling problem was embedded into the problem of deducing adequate Grid resource dimensions. We explained the intricate relation between the workload scheduling and dimensioning problems, but nevertheless were able to propose a manageable linear program combining both problems. To achieve this, we applied several simplifications in order to fit the model into the Grid realm with its large dimensions.

Using this model, we were able to calculate the lambda Grid dimensioning costs for a whole range of operational Grid scenarios. While the base scenarios invariably featured a single overloaded Grid site distributing excess workload, we also paid attention to multi-source scenarios and investigated the additional costs incurred by requiring resilience to single Grid resource failures. These additional costs were shown not to exceed 10% when compared to the base scenario, and these numbers have been verified through an analytical evaluation of regular network topologies with comparable connectivity.

Starting from optimal workload distributions calculated in the off-line combined scheduling and dimensioning problem, we investigated the workload schedules obtained by on-line algorithms using these off-line distributions as resource

target loads. We ensured that the algorithms' complexity did not surpass that of a standard greedy scheduling algorithm. We compared our on-line algorithms to such a greedy strategy and found that, for different cost functions and metrics, the algorithms using the off-line distributions as target show similar results when compared to this greedy algorithm in the scenarios studied.

Several assumptions have been made in this work which can open up future research directions. In particular, for the combined off-line dimensioning and scheduling problem, we have limited ourselves to a single class of jobs (that is, we have described the jobs' input and output demands using a single parameter and have treated these jobs as CBR data sources in our mathematical formulations) while better workload modeling may be possible by distinguishing between relevant job classes. Indeed, our job models have been heavily inspired by the requirements of scientific applications. However, as resource sharing and trans-institute collaboration gain importance in other fields (e.g. broadcasting and media production companies), job models must be adapted appropriately. In media production environments, for instance, applications (e.g. non-linear video editors) have stringent real-time requirements such as low delay, low response times and high network bandwidth. Another case in which our approach may prove to be useful (subject to appropriate job model modifications) is the instantiation and load balancing of software components in component based systems. Such software components do not simply operate on a single specified data set, but rather perform actions on the behalf of multiple concurrent users. As such, these components cannot be modeled as CBR data sources. Instead, an analysis based on queueing theory is necessary to model these components' behavior as a function of their load.

Other assumptions made in this work pertain to the network elements used. In this work, we have assumed optical networks in which cross-connects have unlimited wavelength conversion capabilities. While this fact (combined with our choice of network cost function which does not depend on the number and nature of the wavelength conversions) has helped us to reduce the computational complexity of the combined off-line dimensioning and scheduling problem, a more realistic approach would include limitations on cross-connect capabilities. This observation gains even more importance if we extend our lambda provisioning approach (i.e. a priori setup of long-lived wavelength paths) and move into the realm of schedulable wavelengths, used in application-driven optical networks where applications can (through a unified API) dynamically request temporary end-to-end wavelength paths being set up and torn down.

At last, it can be noted that the Grid concept is not only of interest to a select group of users launching similar applications (as can be the case with research and media production Grids) but is bound to appeal to the general end user as well. These so-called consumer Grids give rise to additional complexities and

questions. Issues that need to be addressed for this kind of Grid include the choice of a suited optical transport technology, job encoding and encapsulation schemes and distributed job routing and scheduling schemes.

The issues raised here are the subject of ongoing research at the IBCN research group, and relevant results will be communicated in future research reports.

A

A Performance Oriented Grid Monitoring Architecture

S. De Smet, P. Thysebaert, B. Volckaert, M. De Leenheer, D. De Winter, F. De Turck, B. Dhoedt, P. Demeester

published in the Proceedings of the 2nd IEEE Workshop on End-to-End Monitoring Techniques and Services (E2EMON), San Diego, 2004, pp. 23-28

Abstract *Resource state monitoring is a critical component of any Grid Management Architecture, providing Grid scheduler, job/execution manager and state estimation components with accurate information about network, computational and storage resource status. Without up-to-date monitoring information, intelligent scheduling decision making would be a near-impossible task. In this paper we describe a scalable, portable and non-intrusive Grid Monitoring Architecture whose implementation decisions were made with performance in mind. We compare it to well-known Grid monitoring systems, and compare our platform's performance to the Globus MDS2.2 and the Globus 3.2 Web Services Information Service (WS-IS) performance.*

A.1 Introduction

A Grid provides a uniform interface to a collection of heterogeneous, geographically distributed resources. These resources are dynamic in nature (i.e. resources

can join/leave the Grid, hardware failures can occur, etc.) and every resource has its own specific properties and status information. This information can in turn be used by the Grid scheduling entity; in a distributed computing environment, one of the key components necessary to be able to perform effective job scheduling (and thereby improve resource utilization), is a resource monitoring architecture. The requirements for such a monitoring system are in no specific order: efficiency, scalability, portability and extensibility.

These requirements have been recognized by the Global Grid Forum (which acts as the governing body for Grid standardization), and this resulted in the conception of a reference architecture for feasible Grid Monitoring systems, dubbed 'Grid Monitoring Architecture' (GMA [1]).

In this paper, we present a feature-rich, highly extensible yet performance oriented Grid monitoring platform, developed according to the GMA specifications. Important features include configurable caching mechanisms, non-intrusiveness, support for third party sensor plugins and an intuitive GUI offering one-click visualization. We discuss the technology decisions that were made when developing this platform, and compare its performance to the Globus Monitoring and Discovery Service [2].

This paper continues as follows: Section A.2 gives an overview of important related work and highlights the differences with our monitoring system. A high-level description of the constituent components is given in section A.3, while technical decisions made during implementation are discussed in section A.4. Our test results are presented in section A.5, followed by a brief look at future work in section A.6 and conclusions in section A.7.

A.2 Related Work

The Grid Monitoring Architecture, as defined by the GGF, consists of three important components: producers, consumers and a directory service (see figure A.1). The directory service stores the location and type of information provided by the different producers, while consumers typically query the directory to find out which producers can provide their needed event data (after which they contact the producers directly). Producers in turn can receive their event data from a variety of providers (software/hardware sensors, applications, whole monitoring systems, databases, etc.). The GMA does not specify the underlying data models or protocols that have to be used.

Multiple monitoring architectures for distributed computing systems have already been successfully deployed. Not all of them follow the guidelines set by the GMA (e.g. Condor's Hawkeye [3] which does not support a decentralized architecture), and some are geared towards monitoring one single resource type (e.g. Remos [4], focusing on network parameters). Below we present some no-

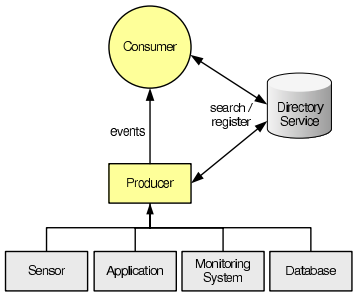


Figure A.1: Grid Monitoring Architecture overview

table Grid monitoring platforms with an architecture similar to our framework, and point out the architectural or implementation-specific differences with our GMA implementation. For a complete overview of Grid monitoring tools we refer to [5].

GMA-compliant Grid monitoring systems include the European DataGrid's Relational Grid Monitoring Architecture R-GMA [6] and GridRM [7]. R-GMA offers a combined monitoring and information system using a Relational Database Management System as directory service and monitoring data repository (this approach offers the possibility to formulate complex queries on the monitored data i.e. it allows to locate monitoring components and retrieve the data they offer using standard SQL statements). The implementation is based on Java servlet technology (using the Tomcat servlet container), trading performance for portability and limited software dependencies.

GridRM is an open source two-layer Grid monitoring framework, the upper layer being structured according to the GMA. This upper layer connects the per-site monitoring systems (the lower layer) in a scalable way. Like R-GMA, GridRM makes use of Java and SQL to query data producers. Currently, GridRM's directory service (containing info on the location of the different resource status providers) can be a bottleneck and/or single point of failure, but work is underway to remedy this problem.

MDS2 is the Globus Toolkit (version 2) Monitoring and Discovery Service, and although MDS development was started before the GMA architectural reference appeared, it can still be seen as a GMA implementation. MDS2 only supports latest-state queries, making it mandatory for the consumers to actively retrieve status information from the GRIS (MDS2 component offering producer-like functionality). In addition, MDS2 does not offer visualization features. In section A.5, we have compared some performance characteristics of MDS2 and our Grid Monitoring Architecture. An extensive comparison of MDS2 against other monitoring frameworks has already been carried out and presented in [8]. It was shown that MDS2 outperforms (i.e. exhibits lower response times and better scalability) the other frameworks mentioned in most use cases. Therefore, we have only compared

our platform's performance to that of MDS2 and its successor, the web services based Information Service [9] from the Globus 3.2 Toolkit. For ease of comparison, we have evaluated this performance using similar tests as described in [8].

A.3 Grid Monitoring Framework Components

In figure A.2 a sample setup of our framework featuring its constituent components is shown. Each component's function is detailed below.

A.3.1 Sensor

Every resource to be monitored has at least one sensor attached to it. Each sensor can monitor different load properties of a single resource. For instance, we have implemented a CPU sensor capable of monitoring CPU load, idle time, frequency and time spent executing user processes. The actual values are then communicated to one or more producers. This list of producers (and conversely, the list of sensors that is allowed to communicate with each producer) and the frequency at which each producer is contacted are readily found in the directory service. The only configurable parameters of the sensor are the location and authentication parameters to query the directory service. When a sensor registers itself with a producer, a permanent data connection (over which load values are pushed) is established, to avoid connection setup overhead on every update. The currently implemented sensors can provide detailed status information on CPU, memory, swap, disk and network usage.

A.3.2 Producer

Producers register themselves with the directory service and publish the type of information (aggregated from the sensors that report to the producer) they provide. This data can be queried by authorized consumers (using a request/reply model) or can be pushed to authorized consumers using a subscription/notification event-based model. In this way, producers correspond to the producer components defined in the GMA. Each producer has its own cache, storing a configurable number of status updates for each registered sensor. Consumers can retrieve either the last known status update from a specific sensor, or historic data from the producer's cache. Furthermore, producers provide a limited number of statistical operations (average, minimum, maximum, standard deviation, etc.) on cached data. Sensor failures (e.g. resource went off-line) can be detected and a failure notification will be sent to consumers who were interested in this sensor's status.

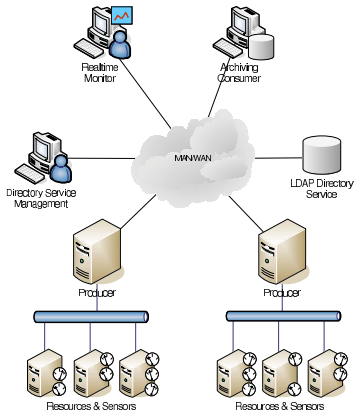


Figure A.2: Grid Monitoring Framework

A.3.3 Directory Service

The directory service contains information on the registered producers (and their respective offered status information), the producer-sensor mappings and producer access control lists. It is queried by producers and sensors to retrieve these mappings, and by consumers to find a set of providers matching some criterium. A web services enabled directory service management component has been implemented.

A.3.4 Consumer

Following the best practices defined by the GMA, consumers query the directory service to find out about producers capable of delivering the desired monitoring data. Consumers proceed by directly contacting these producers, either to retrieve data using a request/reply pattern or to register themselves in order to retrieve future data using an event-based subscription model. Several useful consumers have been implemented, the most important being an archiving consumer (storing data in a relational database and offering a GUI view of historical data), a real-time Java-based visualization agent (see figure A.3) and a Monitoring Server which interfaces directly with Grid Schedulers or Information Services to aid in schedule calculations.

A.4 Technology Analysis

Our GMA-compatible Grid Monitoring Framework was designed to achieve good performance while maintaining a high level of portability.

Our main requirement has driven us to the use of the C++ language in the implementation of the different components. While natively compiled components

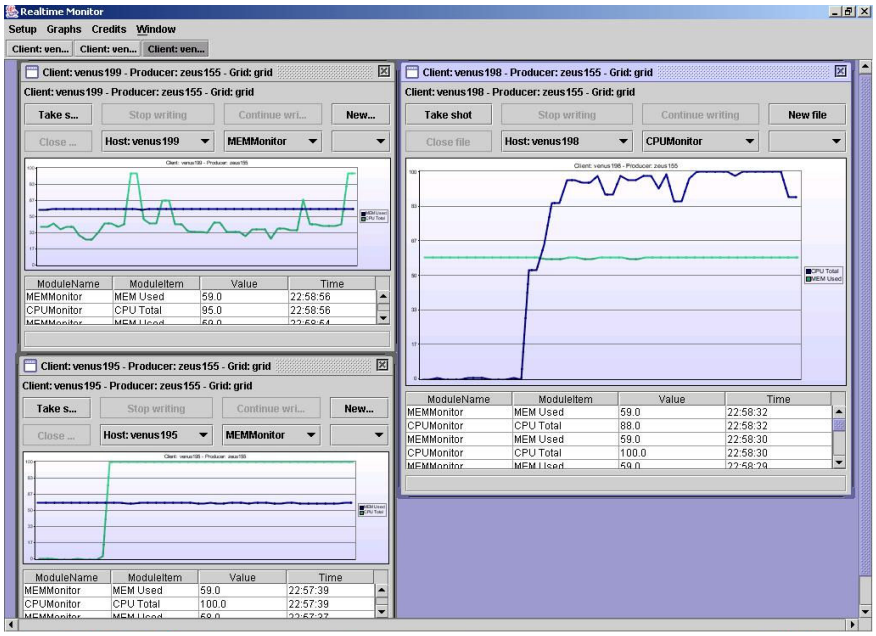


Figure A.3: Real-time Java visualization agent

are known to be more performant than for instance Java bytecode running in a virtual machine, the C++ standard library does not offer cross-platform solutions for various vital application patterns such as networked communication and multiple threads of execution.

Therefore, the need arises to use a portable and performant C++ middleware platform offering elegant implementations of these high-level features. In our monitoring architecture, we have decided to use the Adaptive Communication Environment (ACE [10]). It offers cross-platform multi-threading, and its 'reactor' concept allows for the easy implementation of event-based (including network events) applications. Furthermore, we make use of the 'Acceptor/Connector' pattern offered by ACE to open networked communication channels between producers and sensors.

Whenever a sensor needs to send statistical data to the producers they are registered with, the data is sent in CORBA Common Data Representation (CDR) format, offering a portable, network optimized way of communicating.

The resource monitoring data gathered by the sensors is obtained through the GTop library [11], a portable C/C++ library offering access to performance values related to system resources. Each resource is monitored by a sensor plug-in, which is essentially a shared library. The plug-in approach is enabled by the fact that ACE features cross-platform dynamic loading of shared libraries.

Our directory service is essentially a decentralized LDAP [12] directory server. While the expressive power of LDAP queries does not match that of e.g. an SQL query over a relational database, LDAP allows for performant look-ups in a write-once, read-many context.

Producers provide a WSDL [13] interface through which they are contacted by consumers using SOAP. SOAP is an XML-based RPC protocol which can be transferred over HTTP; as such, the use of SOAP allows for the easy integration of our monitoring architecture in a web services-based Grid environment. In our implementation we used gSOAP as reported on in [14].

An overview of the communication methods used between the various components of our Grid Monitoring Architecture is given in table A.1. It should be noted that SSL encryption is possible for both SOAP-over-HTTP and LDAP communication. This allows access control through the use of user and server certificates. The data updates between sensors and producers can be secured by enabling an SSL socket adapter in these components.

	Sensor	**Producer**	**Consumer**	**Dir. Service**
Sensor	/	ACE	/	LDAP
Producer	ACE	/	SOAP	LDAP
Consumer	/	SOAP	/	LDAP
Dir. Service	LDAP	LDAP	LDAP	/

Table A.1: Communication Technologies

A.5 Results

A.5.1 Testbed Setup

Six machines (AMD Duron 750Mhz, 64MB RAM) have a sensor deployed on them, and four other machines (Intel P4 3GHz, 1GB RAM) carry one producer each; two producers have two sensors registered with them, and the other two have one sensor registered. An OpenLDAP directory server was deployed on a separate machine featuring dual Xeon processors (2.8GHz, 1GB RAM). Lastly, the consumers (implemented as concurrent threads) used in the tests are located on a second dual Xeon machine. All machines are interconnected through a 100Mbps switched Ethernet LAN; this setup allows us to evaluate intrusiveness and scalability of the different components without suffering significant network bottlenecks.

MDS2 was deployed as follows: a GRIS/GIIS pair ran on the Intel P4 machines (instead of the producers), sensors were replaced with GRIS components whose monitoring data was cached by the Intel P4 GIIS. Our LDAP directory service was replaced by a GIIS (on the dual Xeon machines) connected to the lower level

GIISs. Consumers in our MDS tests were spawned from the same machine as our first tests.

The GT 3.2 WS-IS (web services based Information Services) was deployed on the AMD Duron and Intel P4; on the AMD Duron machines, the WS-IS was configured to submit its data to a Pentium 4 machine (which used to run a producer). Again, consumers were spawned from a dual Xeon machine.

A.5.2 Metrics

Two metrics were used to evaluate component performance: *throughput* and *response time*. During a 10 minute period, "users" submitted blocking queries to the component under investigation, while waiting for 1 second between successive queries. The throughput was then taken to be the number of queries handled by the component per time unit; the response time is the average amount of time taken to process 1 user query.

A.5.3 Intrusiveness

The intrusiveness of our producer components is shown in figure A.4, and compared to the load generated by the MDS GRIS. The network traffic generated was monitored using the SCAMPI [15] multi-gigabit monitoring framework (partly developed at our research institution). The CPU load is the average (over the 600 second interval) one minute CPU load average, as measured by *uptime* and related tools.

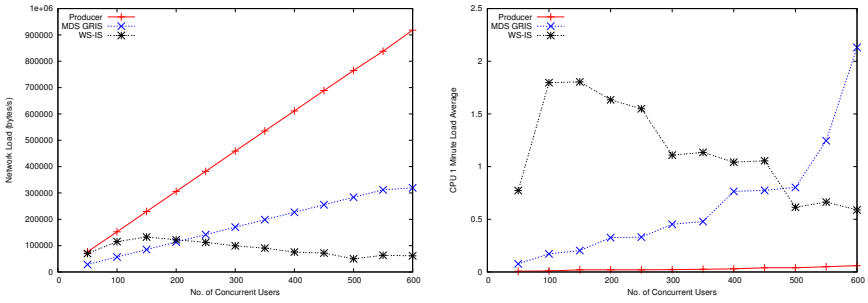


Figure A.4: Producer vs. MDS GRIS vs. WS-IS Network/CPU Intrusiveness

The higher network load generated between our producer and the consumers stems from the use of the SOAP-over-HTTP XML-based communication mechanism (note that we did not enable *zlib* compression). Note that the web services approach used by GT 3.2 imposes a network load comparable to that of our architecture. However, beyond 100 concurrent users, the GT3.2 Information Services

do not scale well, which explains their apparent low network intrusiveness. In analogy, the CPU load generated by the WS-IS seems to degrade with increasing number of concurrent users, but this is slightly deceptive: as the WS-IS does not scale beyond 100 users, it is no longer able to keep up with an appropriate pace of query response generation from this point on. It should be noted that the WS-IS framework is the only monitoring and information framework in these tests which is completely web services based, trading performance for a standards based interface.

A.5.4 Directory Service Scalability

The throughput and response times for directory service queries were compared (figure A.5) to those obtained for queries against the MDS GIIS (operating on cached data). Average response times are lower for our directory service; however, both our directory service and the MDS GIIS don't scale well beyond 450 concurrent users in this scenario on the given hardware. It should be noted, however, that a GIIS typically contains more data (including cached monitoring data) than our directory service, which only stores configuration data, never monitoring data (this should be requested straight from a producer or from an archiving consumer). This led to a bigger result set when the GIIS was queried. In addition, our directory service is a plain OpenLDAP server, without module extensions. We chose not not to show results for the GT 3.2 because of the absence of a dedicated component offering functionality which corresponds to our directory service or the MDS2.2 GIIS.

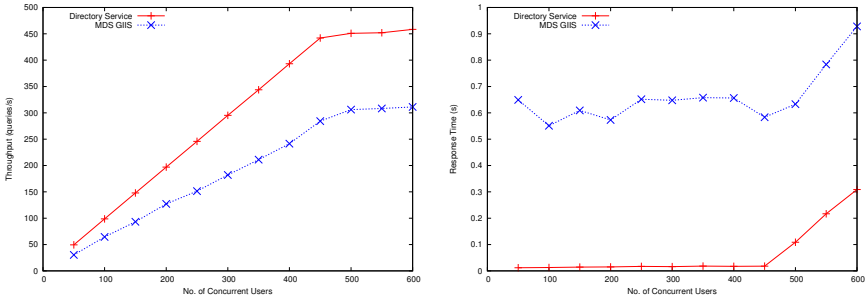


Figure A.5: Directory Service vs. MDS GIIS Scalability

A.5.5 Producer Scalability

In figure A.6, we have compared producer scalability with increasing number of concurrent users for both our framework's producers and MDS GRIS components.

Again, only cached data was requested from the MDS GRIS; due to the use of a push-model our framework's producers always contain up-to-date information, while the GRIS would have to invoke information providers to refresh its data. We measured only small differences between MDS GRIS and our producer (best visible on the response time graph). Beyond 550 concurrent users and using the given machines, MDS GIIS performance started to degrade. We also compared our producer scalability to the scalability of the GT 3.2 Information Service. Again, it is clear that our installation of GT3.2 does not scale well beyond 100 concurrent users (the GT 3.2 results even forced us to use a logarithmic scale in the right part of figure A.5, which shows that response times differ by as much as a factor of 100).

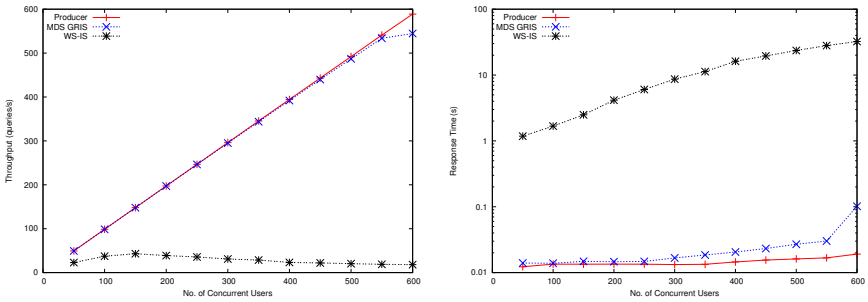


Figure A.6: Producer vs. MDS GRIS vs. WS-IS Scalability

A.6 Future Work

Because manually deploying sensor plug-ins on multiple resources is tedious, our main focus in the near future will be on platform bootstrapping components. The main idea here is that these bootstrapping components are loaded on a resource as soon as this resource is brought on-line ("installed" when the resource is a computer). Their main objective is to record this resource's presence in the directory service, auto-install suitable (based on the resource's class) sensor plug-ins and register with a set of producers (based on the type of monitoring data offered and the resource's location).

A second topic of interest is the development of proxy producers (remember that sensors and producers communicate using ACE socket streams) which allows to collect sensor data from other monitoring platforms' "information providers" and use these data in our framework.

Lastly, an additional effort is required to adapt our implementation (in particular the interfaces implemented by the various components) in order to comply

with the Web Services Resource Framework [16] (unifying Grid and Web Services communities) specifications.

A.7 Conclusions

We presented a performant and portable implementation of the GGF Grid Monitoring Architecture. Performance was obtained through the use of C++ as base implementation language; portability then dictated the use of appropriate middleware for which we chose the Adaptive Communication Environment. We compared the performance of our implementation to that of the Globus MDS2 system and its successor (the GT 3.2 web services based Information Service using the default supplied OGSI-compliant container), with good results in terms of throughput and response time, both for producers and the directory service. Multiple ready-to-use consumers (including real-time visualization) have been implemented. Within our implementation, heavy use is made of SOAP for consumer-producer communication. With the apparent convergence of the web services and grid communities in mind, we expect this to ease the deployment of our implementation in a services enabled grid environment.

Acknowledgment

B. Volckaert would like to thank the Institute for the Promotion of Innovation through Science and Technology in Flanders.

References

[1] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. *A Grid Monitoring Architecture*. `http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf`, 2002.

[2] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. *Grid Information Services for Distributed Resource Sharing*. In Proc. of the 10th IEEE International Symposium on High-Performance Distributed Computing, pages 181–194, 2001.

[3] *HawkEye: A Monitoring and Management Tool for Distributed Systems*. `http://www.cs.wisc.edu/condor/hawkeye/`.

[4] Bruce Lowekamp, Nancy Miller, Thomas Gross, Peter Steenkiste, Jaspal Subhlok, and Dean Sutherland. *A resource query interface for network-aware applications*. Cluster Computing, 2(2):139–151, 1999.

[5] M. Gerndt, R. Wismuller, Z. Balaton, G. Gombas, P. Kacsuk, Zs. Nemeth, N. Podhorszki, H. L.Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef. *Performance Tools for the Grid: State of the Art and Future*. Research Report Series, Lehrstuhl fuer Rechnertechnik und Rechnerorganisation (LRR-TUM) Technische Universitaet Muenchen, 30, 2004.

[6] A. Cooke, A.Gray, L. Ma, W. Nutt, J. Magowan, P. Taylor, R. Byrom, L. Field, S. Hicks, and J. Leake et al. *R-GMA: An Information Integration System for Grid Monitoring*. In Proc. of the 11th International Conference on Cooperative Information Systems, pages 462–481, 2003.

[7] M.A. Baker and G. Smith. *GridRM: A Resource Monitoring Architecture for the Grid*. In Springer-Verlag, editor, Proc. of the 3rd International Workshop on Grid Computing, pages 268–273, 2002.

[8] X. Zhang, J.L. Freschl, and J. Schopf. *A Performance Study Of Monitoring and Information Services for Distributed Systems*. In Proc. of the 12th IEEE International Symposium on High-Performance Distributed Computing, pages 270–282, 2003.

[9] *WS Information Services website*. `http://www-unix.globus.org/toolkit/docs/3.2/infosvcs/ws/key/index.html`.

[10] D.C. Schmidt and S.D. Huston. *C++ Network Programming: Mastering Complexity Using ACE and Patterns*. Addison-Wesley Longman, 2002.

[11] *LibGTop website*. `http://www.gnu.org/directory/libs/LibGTop.html`.

[12] Wahl M., T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3)*. IETF RFC 2251, 1997.

[13] *Web Service Definition Language*. `http://www.w3.org/TR/wsdl`.

[14] R.A. van Engelen and K.A. Gallivan. *The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks*. In CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, page 128, 2002.

[15] J. Coppens, S. Van den Berghe, H. Bos, E.P. Markatos, F. De Turck, A. Oslebo, and S. Ubik. *SCAMPI: A Scalable and Programmable Architecture for Monitoring Gigabit Networks*. In Proc. of the 1st Workshop on End-to-End Monitoring Techniques and Services, LNCS 2839/2003, pages 475–487, 2003.

[16] K. Czajkowski and al. *The WS-Resource Framework Version 1.0*. `http://www.globus.org/wsrf/specs/ws-wsrf.pdf`.

B

Network Aspects of Grid Scheduling Algorithms

P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester

published in the Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems (PDCS), San Francisco, 2004, pp. 91-97

Abstract *Computational Grids consist of a multitude of geographically distributed resources. The co-allocation of several of those resources allows for the execution of highly computing-intensive and data-intensive jobs. In order to obtain quality schedules (in terms of job response time and resource utilization), different factors such as resource load (both computational resource load and bandwidth usage on the interconnecting network) and data location need to be taken into account. We use a discrete-event simulator to accurately model the network interconnecting the different Grid sites, and study the scheduling of both data-intensive and CPU-intensive jobs (which can use different data transfer strategies over this network) on a Grid. In particular, the scenarios of interest that we study for different network bandwidths include the use of simultaneous data transfer and job execution (vs. data pre-staging), and the use of capacitated VPNs to secure up-front guaranteed resource-to-resource bandwidth availability for certain job types (vs. the use of a pure FCFS policy for the setup of data connections). Our results*

show that average job response time and resource reservation accuracy can be improved by including network information in traditional scheduling strategies, and even more by preventing discrimination of certain job types by making upfront bandwidth reservations (VPN) for those types.

B.1 Introduction

A fairly recent evolution in the domain of distributed computing is the concept of Computational Grids [1]. Such a grid consists of the aggregation of a dynamically changing set of heterogeneous resources, scattered over several locations. These include computational resources, storage resources, data generating instruments and the interconnecting network. The aggregate power of a grid is suited to handling resource-intensive jobs.

Selecting a set of resources and allocating them to a job is the task of the scheduling mechanism. Due to substantial data sizes involved in job execution, and limited network bandwidth availability between resources (when compared to e.g. a local cluster), this resource selection needs to take into account network load and data locality. Indeed, as input data needs to be present at the execution site at the time it is needed, available bandwidth (or lack thereof) determines the actual starting time of a job when all its input data is pre-staged to its allocated computational resource; if the job accesses its input data in a pure sequential block-per-block way, job execution may start in parallel with the data transfer. In the latter case, however, the job's computational progress rate (which relates to the idle time left on the computational resource if this were the only job executing) is also limited by that very bandwidth.

In addition, when dealing with different job or service classes, resources can be connected in a VPN offering per-service bandwidth guarantees, or all resource-to-resource connections can be setup on demand without service discrimination.

For both CPU-intensive and data-intensive jobs, we investigate these different data access and connection setup patterns for different network scenarios. Using NSGrid [2], our in-house developed Grid simulator built on top of ns-2 [3], we compare schedules produced by different algorithms (taking into account computational resource load, network bandwidth and job requirements) in terms of job response time and resource utilization.

This paper is structured as follows: in section B.2 we give an overview of related work. Section B.3 details the simulation models that were used: Grid, Network, Computational/Storage/Information Resource and Job models. Section B.4 elaborates on the different Grid scheduling strategies used, while the evaluation of those strategies (using the NSGrid simulation environment) for different job classes in a typical Grid topology is detailed in section B.5. We briefly address future work in section B.6. Finally, section B.7 summarizes the paper and gives

some concluding remarks.

B.2 Related Work

Our Grid simulation environment (NSGrid) is based on the well-known ns-2 [3] network simulator. While not providing the most scalable simulation kernel (more scalable (C++) simulation frameworks are available, such as *DaSSF* [4] and *OMNeT++* [5]), ns-2 is an up-to-date network-oriented simulator providing models for a wide range of protocols in all layers of the network stack.

Notable existing Grid simulators include Bricks, MicroGrid, SimGrid and GridSim.

The Bricks Simulator [6] focuses on client/server interaction in global high performance computing systems. It allows for a single centralized scheduling strategy, which does not scale well to large Grid systems and does not support the notion of multiple (competing) schedulers.

MicroGrid [7] is an emulator modelled after Globus, allowing for the execution of Globus-enabled applications on a virtual Grid system. Research into the area of Grid scheduling algorithms can be cumbersome with this kind of approach, since it requires the construction of an actual application to test.

SimGrid [8] is designed to simulate task scheduling (centralized or distributed) on Grids. Version 1 of SimGrid can be regarded as a low-level toolkit (which interfaces to the C programming language) from which domain-specific simulators can be built. The second version of SimGrid is dubbed *MetaSimGrid* and is essentially a simulator built upon this toolkit to enable the construction of simulation with multiple schedulers (as C programs). Models for network links as well as for TCP connections are present in SimGrid. This validated TCP implementation allows for smaller simulation times when compared to the packet-level TCP simulation performed by network simulators. Of course, simulations using other transport protocols that are not readily available in SimGrid require that these protocols are implemented first, whereas using a network simulator ensures easy access to a wide range of protocols. The simulated application consists of several tasks, organized into a Directed Acyclic Graph (DAG). MetaSimGrid is focused on scheduling this application type in a master-slave environment.

GridSim [9] is a discrete-event Grid simulator based on JavaSim. This simulator allows to simulate of distributed schedulers, and is specifically aimed at simulating market-driven economic resource models. While its computational resource models are highly configurable, only a basic notion of network connectivity is supported and underlying network dynamics are not accurately simulated.

Scheduling jobs over multiple processing units has been studied extensively in literature. Machine scheduling [10] [11] is concerned with producing optimal schedules for tasks on a set of tightly-coupled processors, and provides analytical

results for certain objective functions. Jobs are commonly modelled as task graphs, or as continuously divisible work entities. As these models do not deal with "network connections" or "data transfers", they do not capture all the Grid-specific ingredients described in the previous section.

In [12], multi-site execution of divisible jobs is discussed. Jobs can be split into (communicating) subjobs which are then executed simultaneously on different computational resources. The network over which the subjobs communicate is not modelled directly; rather, it is assumed that the network's influence (bandwidth, delay) on the job's run time can be modelled by a single "overhead" parameter.

A similar job model is used in [13] and [14]. Here, the allocation of processors to rigid parallel applications on a purely space-shared (multi)cluster system is studied. Applications consist of a number of possibly communicating jobs, to be executed in parallel. Each job requires exactly one processor, which it occupies exclusively during its execution (i.e. no time-shared processors). Figures for the fraction of idle processors at a given point in time are deduced using statistical techniques, while the influence of a slow inter-cluster communication network is incorporated entirely in a slowdown factor α. This contrasts with our approach, as we study applications consisting of non-communicating jobs, each of which can be executed on a single *time-shared* processor. In addition to computational resources (i.e. clusters), we also treat other resources such as data storages explicitly. The figures relevant to our scheduling scenarios are obtained through simulation in which network traffic (both inter-cluster and intra-cluster) is simulated to the packet level.

Simulation of Grid scheduling strategies which take both computational resources and data resources (more specifically, data locality) into account have been reported upon in [15]. In this work, however, the network connecting different sites is not simulated, but it is assumed that the different sites are connected through a VPN-like construction over which TCP communication occurs. Scenarios where files are pre-staged are considered, but data transfers in parallel with job execution are not.

Replication optimization on an operational Grid (the EU DataGrid [16]) has been studied in [17]. Again, only pre-staged data scenarios are considered. The use of capacitated VPNs for different job classes, as well as live remote data access are not covered in this work.

Distributing work packets for collaborative computing efforts (e.g. SETI [18], MCell [19]) to computational elements is discussed in [20]. Because of the application's particular nature, the grid can be modelled as a tree, with all work packets originating from the root node. In our model, jobs can be submitted by users residing at different sites and may need data not present at their submission site.

The algorithms we study do not discriminate upon a job's internal characteristics or structure; that is, we do not engage in application-level scheduling such as

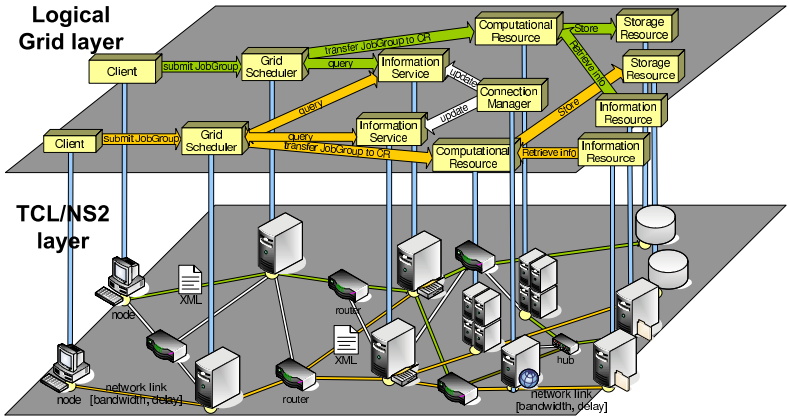


Figure B.1: NSGrid architecture

provided by *AppLeS* [21] . Using the latter approach, one separate scheduler needs to be constructed per application type. NSGrid allows the simulation of multiple scheduling scenarios, including those using a single centralized schedule as well as those having multiple competing schedulers (not necessarily one per application type).

B.3 Simulation Models

B.3.1 Grid Model

In our simulation environment, Grids are modelled as a collection of interconnected and geographically dispersed *Grid sites*. Each Grid Site can contain multiple *resources* of different kinds (their modelling is explained below in more detail) such as Computational Resources(CR), Information Resources(IR), Storage Resources(SR) and network resources. A key property of this model is the explicit treatment of the network as a "resource", allowing the scheduler to take decisions based on observed and expected future load of the network interconnecting the different processing/information/storage elements.

At each Grid Site, resource properties and status information are collected in a local *Information Service*. Jobs are submitted through a *Grid Portal* and are scheduled on some collection of resources by a *Scheduler*. To this end, the scheduler makes *reservations* with the *Resource Managers*; in our environment, a *Connection Manager* manages a collection of network links, while the *Computational*, *Information* and *Storage Resources* double as their own manager. All these entities are attached to their own ns-2 node in the underlying simulated network (recall that NSGrid is built on top of ns-2). LAN links interconnect a Grid Site's local re-

sources, while Grid Sites themselves are interconnected by means of MAN and WAN links.

B.3.2 Network Model

Interconnections between resources are modelled as a collection of point-to-point (i.e. between two non-network resources) connections, each offering a guaranteed total bandwidth available to Grid jobs. Of course, these connections can only be set up if, in the underlying ns-2 network topology, a route (with sufficient bandwidth capacity) exists between the nodes to which these resources are attached. Grid resources can also be interconnected by means of capacitated VPNs: in this case, a VPN tunnel (with guaranteed bandwidth availability) is set up between two Grid resources for a particular Grid job service type. This VPN tunnel carries all connections matching the VPN's endpoints and service type. Such connections can be setup as long as the VPN's residual bandwidth can satisfy the connection demands. VPNs allow for the upfront reservation of bandwidth for a particular (prioritized) service type. The Grid scheduling component we implemented bases its calculations on these connections' bandwidths and makes reservations with resources in such a way that an average throughput for each connection equal to its "guaranteed" bandwidth is obtained. This means that the produced schedules are also correct w.r.t. network bandwidth usage, even if in reality, a network management infrastructure allowing the reservation of end-to-end connections with guaranteed bandwidth is not available.

B.3.3 Computational Resource Model

Each Computational Resource is viewed as a monolithic entity with a certain processing power. Its capabilities are defined by the following parameters:

- The number of processors and their respective processing power and memory
- Load: job load on the Computational Resource

This model can be used to represent both multiprocessors and clusters, provided that, in the latter case, the internal network connecting the various cluster nodes performs sufficiently (i.e. the network interconnecting the various processing elements - which is not modelled - never becomes a performance-limiting bottleneck). If the Computational Resource is time-shared, rigid portions of a processor's power can be allocated to individual jobs.

B.3.4 Information/Storage Resource Model

Information Resources and Storage Resources serve the purpose of providing input data for jobs, and providing disk space to store output data respectively. In our model, Storage Resources are described by

- The total available storage space
- Load: Storage space allocated to jobs

Information Resources on the other hand are described by the data sets (and their respective size) available as input for a job. While an Information or Storage Resource does not perform computational work, it can be attached to the same network node as some Computational Resource.

B.3.5 Job Model

The atomic (i.e. that which cannot be parallelized) unit of work used throughout this paper is coined with the term *job*. Dependencies between individual jobs (described by a Directed Acyclic Graph) can be expressed using the notion of *Jobgroups*. Each job is characterized by its length (time to execute on a reference processor), its required *input data sets*, its need for *storage* (used for output data), and (if the jobs processes and/or produces data in sequential blocks) the *burstiness* with which these data streams are read or written. During a job's execution, a certain minimal computational progress is to be guaranteed at all times (i.e. a deadline relative to the starting time is to be met).

Knowing the job's total length and the frequency at which each input (output) stream is read (written), the total execution length of a job can be seen as a concatenation of instruction "blocks". The block of input data to be processed in such an instruction block is to be present before the start of the instruction block; that data is therefore transferred from the input source at the start of the previous instruction block. Similarly, the output data produced by each instruction block is sent out at the beginning of the next instruction block. We assume these input and output transfers occur in parallel with the execution of an instruction block. Only when input data is not available at the beginning of an instruction block or previous output data has not been completely transferred yet, a job is suspended until the blocking operation completes. A typical job execution cycle (one input stream and one output stream) is shown in figure B.2. The presented model allows us to mimic both *streaming* data (high read or write frequency) and *data staging* approaches (read frequency set to 1).

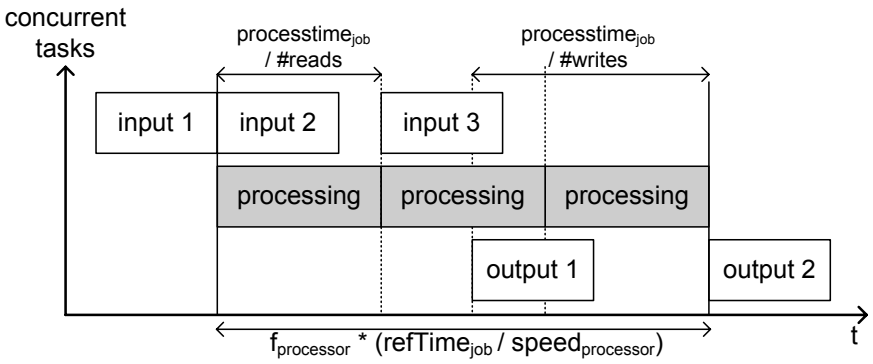


Figure B.2: Non-blocking job, simultaneous transfer and execution

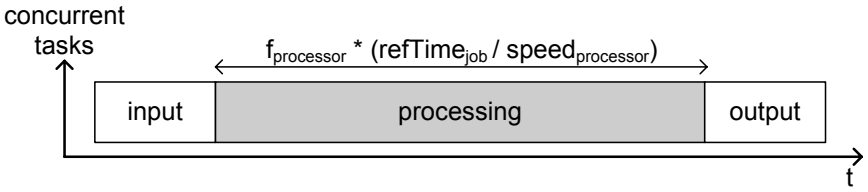


Figure B.3: Non-blocking job, pre-staged input data

B.4 Scheduling Algorithms

When jobs are submitted, a Scheduler needs to decide where to place the job for execution. The scheduling algorithm used in making this selection has a big impact on Grid performance, and influences overall Grid job throughput, Grid resource efficiency etc. If the scheduler is unable to allocate the needed resources for a job, the job gets queued for reschedulement in the next scheduling round. The time between two scheduling rounds can be fixed, but it is also possible to set a threshold (e.g. time limit or number of unscheduled jobs in the queue) which triggers the next scheduling round. In what follows we will explain the different scheduling strategies used in our simulations. During each scheduling round, every algorithm processes submitted yet unscheduled jobs in a FCFS fashion, and attempts to minimize the completion time for each job. Once scheduled, our scheduler does not attempt to pre-empt jobs.

All jobs run on a single processor; processors can be time-shared (i.e. serve multiple jobs simultaneously by allocating portions of its processing power to each such job). Intelligent allocation of these portions to jobs is necessary to prevent jobs from blocking (i.e. wasting CPU cycles allocated to it) when they depend on bandwidth-limited remote data access, as described in the previous section. Jobs

can expose dependencies (described by directed acyclic graphs (DAG)), but are taken not to communicate with each other.

Below we have presented the algorithms we used in our study; all are *queueing* algorithms [22], that is, whenever an algorithm is invoked, it will attempt to schedule the not-yet scheduled jobs in the order of arrival on the time-shared resources. Jobs that cannot be scheduled will be requeued, preserving the relative order of arrival. Since the jobs are never pre-empted, the use of queueing algorithms when scheduling jobs on time-shared resources brings with it the risk to allocate minimal leftover time shares to certain jobs, introducing long turnaround times. Therefore, during the simulations we have demanded that a single processor (note that a single Computational Resource can contain multiple processors) be time-shared by no more than 3 concurrent jobs in order to avoid degenerate fragmentation. As the goal of each algorithm is the minimization of each job's response time, a natural metric to benchmark the different algorithms is the average job *turnaround time*. This metric is discussed in more detail in subsection B.5.5.

B.4.1 Network Awareness

B.4.1.1 Non-Network Aware Scheduling

Non-Network aware scheduling will compute Grid job schedules based on the status of the Computational, Storage and Information Resources (as provided by the Information Services). Algorithms that use this kind of approach will not take into account information concerning the status of resource interconnections. The decision of which resources to use for a job will be based on the information acquired from the different Information Services (i.e. job execution speed and end time will be calculated based on the status of CR/IR/SR). It is precisely because Non-Network aware algorithms assume that residual bandwidth on network links is "sufficient", that jobs can block on I/O operations: their computational progress is no longer determined by the Computational Resource's processor fraction that has been allocated to it (which, together with the job's length and the Computational Resource's relative speed determines its earliest end time *if all input and output transfers complete on time i.e. before the start of the appropriate instruction block*), but rather by the limited bandwidth available to its input and output streams. Note that the fact that network information is discarded during the scheduling implies that no connection reservations (providing guaranteed available bandwidths) are made - these would allow to accurately predict the job's running time.

We have used non-network aware scheduling as a naive heuristic to compare the improved (network-aware) algorithms to in our simulations.

B.4.1.2 Network Aware Scheduling

Network aware scheduling algorithms will not only contact the Information Services (for information about resources that adhere to the job's requirements), but will also query the Connection Manager for information about the status of the network links interconnecting these resources (i.e. the Connection Manager will send the Grid Scheduler information about connections that can be set up between IR/CR couples (necessary for job input retrieval) and CR/SR couples (needed for job output storing)). Based on the answers from the Information Services and Connection Manager, the scheduling algorithm is able to calculate job execution speed and end time more accurately, taking into account the speed at which input/output can be delivered to each available Computational Resource. For jobs with 1 input stream and 1 output stream, the best resource (CR/IR/SR) triplet is the one that minimizes the expected completion time of the job. This value is determined by the available processing power to that job on the Computational Resource (and its relative speed), the job's length, the job's total input and output data size and the residual bandwidth on the observed links from IR to CR and from CR to SR:

$$Duration_{job} = \min_{Resources, Conn} (duration(job, Resources, Conn))$$

As explained, for some (CR,SR,IR) triplet, due to bandwidth constraints, this duration may be significantly higher than the value calculated from the job's length and the CR's relative speed, even if job execution and data transfer occur simultaneously. The scheduler selects the optimal CR/IR/SR triplet and contacts the central Connection Manager to perform the necessary connection setups (the necessary bandwidth of these connections is calculated by the scheduler). The job then gets transferred to the selected CR for processing and input/output is sent from/to the IR/SR over the reserved connections. If no (local or remote) Resources satisfying the job's requirements can be found, or if no connections with sufficient bandwidth are available, the job will be queued and prepared for reschedulement. The time it takes for a job to complete since it has been submitted by the client can be broken up into:

- sending the job to the scheduler
- time spent in the scheduler's queue
- time needed for the co-allocation of resources (including network resources) allocated to that job
- transfer time for the first input data block(s)
- time needed to process the job at its allocated execution speed
- transfer time for the last output data block(s).

Each of these can be found in figure B.4. Note that no job can become blocked because reservations are made with network resources, excluding the network from becoming an unexpected bottleneck.

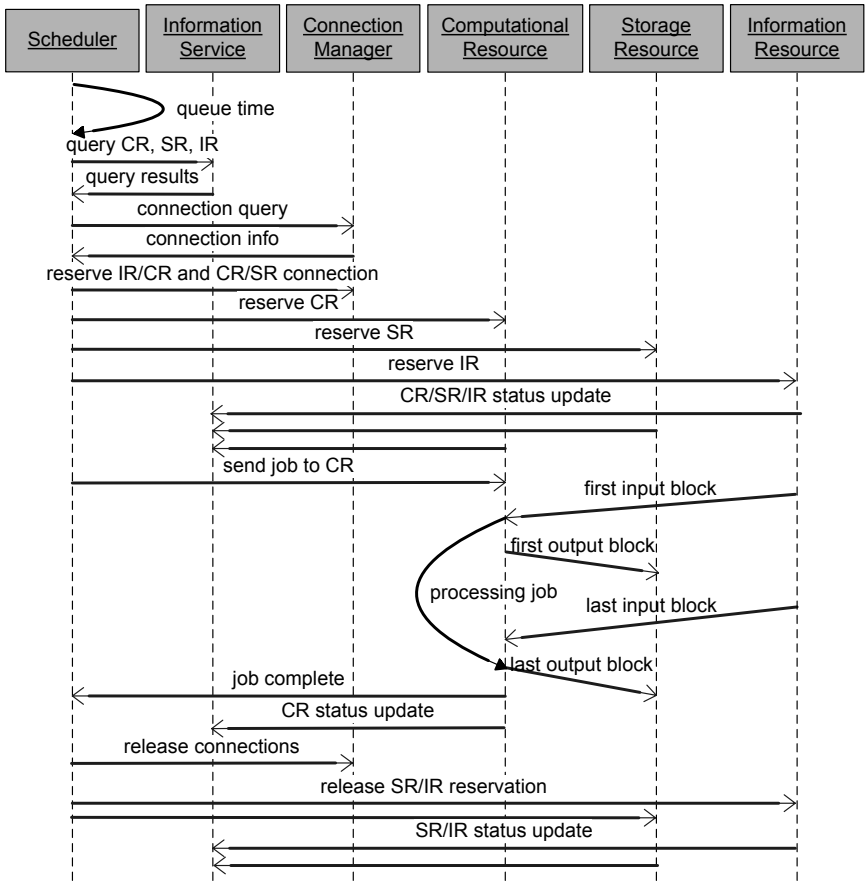


Figure B.4: Job scheduling (Network Aware)

B.4.2 Resource Locality Preference

B.4.2.1 PreferLocal Scheduling

PreferLocal scheduling algorithms attempt to place a job on a site's local Computational/Storage/Information Resources for processing, as we believe that, from an economic viewpoint, it can be assumed that remote resources are only used when necessary. When local processing is impossible (either because the job's requirements cannot be met locally, because the maximum computational load has

been reached, or because I/O requirements are not met), the scheduler looks at the status of the remote resources (received from the different Information Services) and, if possible, selects a Computational/Storage/Information Resource triplet (not necessarily all residing at one particular Grid site) meeting the job's requirements and prefers the triplet which allows for the fastest job end time (this job end time can be calculated in both a network aware or a non-network aware fashion). The job is then transferred to this remote Computational Resource for processing and I/O is sent from/to the selected Information and Storage Resource. If no (local or remote) resources satisfying the job's requirements can be found, the job gets queued for rescheduling during the next scheduling round.

B.4.2.2 Spread Scheduling

Spread algorithms schedule resources the same way as PreferLocal algorithms, with the exception that they do not prefer resources local to the job's originating site.

B.5 Simulation Results

B.5.1 Simulation Environment

All simulations were performed on an OpenMosix cluster with 20 nodes. Each node contains an AMD Athlon XP1700 processor and 1 GB RAM, and runs Debian GNU/Linux (Woody). The average time to complete a single simulation scenario was about 48 hours. The exact inputs to the simulator are described below.

B.5.2 Simulated Topology

A fixed Grid topology was used for all simulations presented here. First, a Wide-Area Network (WAN) topology (containing 8 core routers with an average out-degree of 3) was instantiated using the *GridG* tool [23]. Amongst the edge LANs of this topology, we have chosen 12 of them to represent a Grid site (each having its own Computational, Storage and Information Resource). Furthermore, we have homogenized the capacities of each WAN link, which we then treated as a parameter in our simulations. Each site has its own *Information Service* (storing Resource properties and status) and local *Grid portal* (through which users can submit jobs). Local resources are connected through 1Gbps LAN links.

B.5.3 Job parameters

We have used two different job types in our simulations; one is more data-intensive (i.e. higher data sizes involved), while the other is more CPU-intensive. At each

Grid Site, two "clients" have been instantiated, one for each job type. Each client submits mutually independent jobs to its Grid Portal using a uniform interarrival time distribution. All jobs need a single IR and a single SR. The ranges between which the relevant job parameters vary have been summarized in table B.1. Both job types make up 50% of the total job load; in each simulation, the job load consisted of 1200 jobs.

	CPU-Job	**Data-Job**
Input(GB)	0.01-0.02	1-2
Output(GB)	0.01-0.02	1-2
IAT(s)	100-200	100-200
Ref. run time(s)	400-1200	200-400

Table B.1: Relevant job properties

For each scheduling algorithm, we have chosen to use a fixed interval of $50s$ between consecutive scheduling rounds. From the arrival rates in table B.1 and the fact that multiple sites submit job simultaneously, it follows that we are likely to find multiple jobs in the queue at the start of each scheduling round.

B.5.4 Resource dimensions

B.5.4.1 Computational Resources

We have assigned one Computational Resource to each Grid Site. To reflect the use of different tiers in existing operational Grids [16], not all Computational Resources are equivalent: the least powerful CR has two processors (which operate at the reference speed). A second class of Computational Resources has four processors, and each processor operates at twice the reference speed. The third - and last - Computational Resource type contains 6 processors, each of which operates at three times the reference speed. Conversely, the least powerful type of CR is three times as common as the most powerful CR, and twice as common as the middle one. It is assumed that all processors can be time-shared between different jobs.

B.5.4.2 Storage Resources

For the simulations performed, we have focused on determining the influence of the use of network resource status on the schedule calculation; therefore, we have assumed that Storage Resources offer "unlimited" disk space that can be read and written at "sufficiently high" speed (i.e. higher than the needed data transfer bandwidths). Each site has at its disposal exactly one such Storage Resource.

B.5.4.3 Information Resources

Each site's Information Resource contains 6 out of 12 possible data sets. These data sets are distributed in such a way that 50% of the jobs submitted to a site can have local access to its needed data set.

B.5.5 Performance Metrics

B.5.5.1 Average Job Response Time

We define the *response time* of a job as the difference between its end time and the time it is submitted to the scheduler. In figure B.5 we present this average

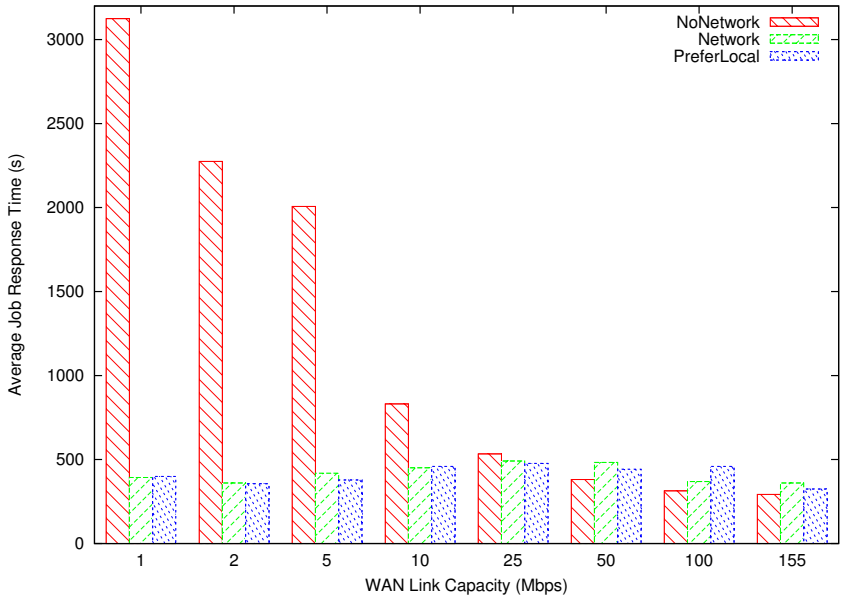


Figure B.5: Average Job Response Time

job response time for the three algorithms we discussed earlier. In this particular simulation, simultaneous execution and data transfer were allowed; data connections were setup on a FCFS basis without upfront VPN dimensioning. Clearly, for low bandwidths, not taking into account network status (when computing the schedule) incurs a severe penalty; when bandwidth grows, the importance of this network information degrades (when the job load is constant) as the network no longer creates a bottleneck. In fact, for high bandwidths, it is possible for the non-network aware algorithm to perform slightly better; this is due to the conservative nature of our network-aware algorithms. For instance, they take for granted that

the maximum data transfer rate is only 95% of the available bandwidth (i.e. 5% protocol overhead) and adjust their allocations accordingly.

In our simulations, no improvement is obtained from preferring local resources. Intuitively, we expected this latter strategy to create better schedules for data-intensive jobs (as intra-site network links have high capacities). However, this improvement is neutralized by the asymmetry of the Computational Resources: jobs submitted at a site containing a slower Computational Resource, are now less likely to be executed on a faster one (which is of course the case if the best resource collection is selected for a job).

B.5.5.2 Computational Resource Idle Time

If job execution and data transfer occur simultaneously, jobs can block (i.e. induce idle time on their time-shared Computational Resource within the processing power fraction allocated to that job) if it needs to wait for input data to arrive. This scenario is plausible when a non-network aware scheduling algorithm is used: while available network bandwidth (in particular, between the job's Computational Resource and the Information Resource providing it with input data) influences the minimum duration of a job on that Computational Resource, these algorithms do not take into account this bandwidth. This results in possible overallocation of the time-shared Computational Resource: a fraction of the resource is reserved uniquely for this job, but the job is unable to exploit the computing power allocated to it to its full extent. This means that -within its allocated fraction- a job induces idle time on the Computational Resource. Again, the incurred penalty grows with lower bandwidths. Figure B.6 shows the amount of idle time created by the non-network aware algorithm in such cases.

In contrast, the network-aware scheduling algorithms we discussed will be able to "tune" their Computational Resource allocations with network bandwidth in mind, to ensure that no Computational Resource is unnecessarily left idle.

B.5.5.3 Influence of sequential data processing

In figure B.7, we have replotted the average job response time for the same job load; now, however, jobs were not able to start executing while still downloading data (i.e. pre-staging of the entire input to the execution site was required). As the execution/transfer parallelism is lost, average response times for network-aware algorithms increase. However, this loss of parallelism does not influence the relative behavior of the different algorithms (network aware or not) discussed before. For low bandwidths, the network-unaware algorithm produces better response times when pre-staging data, as the jobs cannot block during execution in this case.

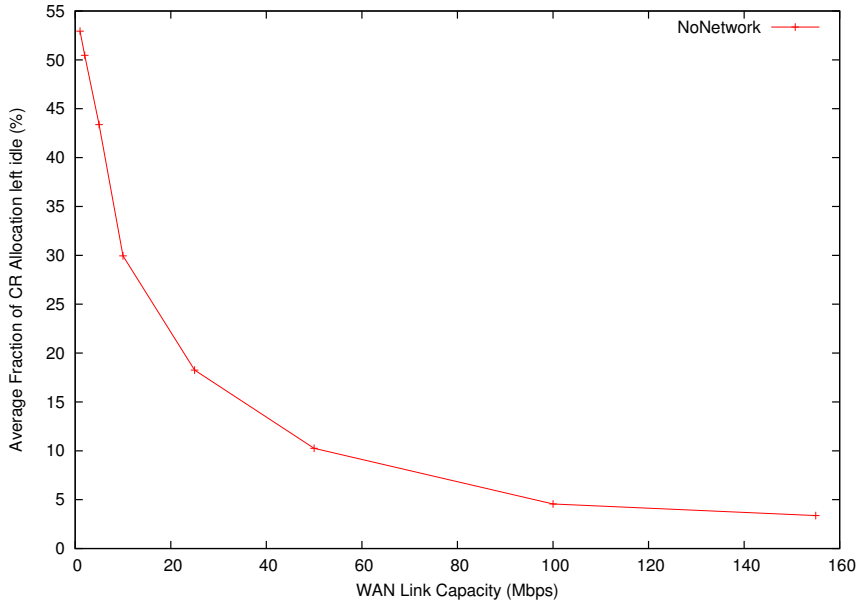


Figure B.6: CR Allocations: Idle Time

B.5.5.4 Influence of capacitated VPNs

If data connections are setup on demand using a pure FCFS scheme, it is likely that data-intensive jobs will quickly use up all of the available bandwidth, causing CPU-intensive jobs to remain queued for a longer period of time.

The upfront reservation of bandwidth to each job type ensures that these CPU-intensive jobs will never be excluded from remote execution (i.e. move towards a faster Computational Resource). We have simulated our job load again, but this time we have setup VPNs for the two job classes (data-intensive vs. CPU-intensive). We reserved more bandwidth for the data-intensive jobs (about $20\% - 80\%$ ratio). The job response time for the different algorithms in this scenario is shown in figure B.8. This approach visibly improves the response time: cpu-intensive jobs do not remain queued for an extraordinary period of time, and as these jobs have high run times, this has a significant impact on the average job response time.

We believe further improvement is possible if bandwidth is distributed more intelligently across the different job classes in a way that takes into account their respective processed data sizes and run times, but at the time of this writing, we have not yet pursued this idea any further.

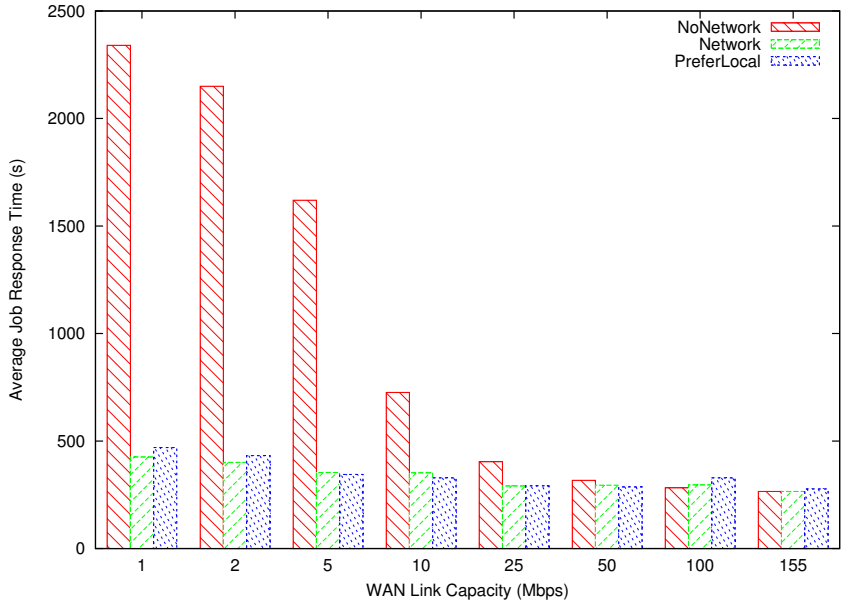


Figure B.7: Response Time: pre-staged input

B.6 Future work

We plan to investigate the quality of schedules produced by the network aware algorithms under different Information Resource data set replication strategies. In addition, the constraint concerning the homogeneity of the inter-site WAN bandwidth will be relaxed. Besides evaluating the existing algorithms for more elaborate Grids, we also plan to evaluate more refined network aware algorithms that can take into account a job's service class, priority and computation/communication ratio. Due to service class differences, we believe that the best results can be obtained by applying algorithms specifically tailored to schedule all jobs in one particular service class.

B.7 Conclusions

In this paper we have investigated the use of network status information when calculating schedules for a Grid. Whether data is pre-staged or accessed remotely during the job's execution, this information allows to create significantly better schedules in terms of both job response time and Computational Resource idle time. From our simulations, it follows that upfront reservation of bandwidth (between

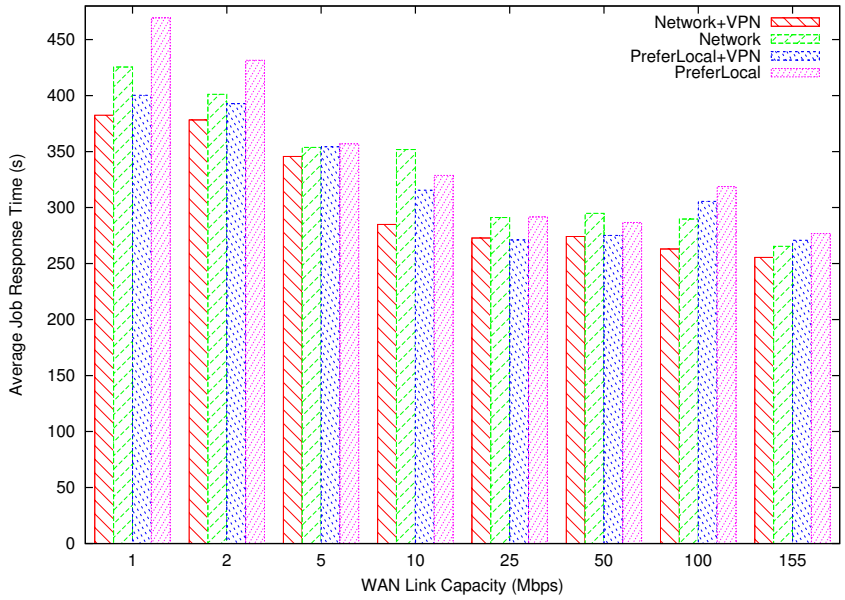


Figure B.8: Response Time: VPN reservations

Grid resources) for certain job types can improve the response time by avoiding that data-intensive jobs monopolize available bandwidth.

References

[1] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[2] B. Volckaert, P. Thysebaert, F. De Turck, P. Demeester, and B. Dhoedt. *Evaluation of Grid Scheduling Strategies through a Network-aware Grid Simulator*. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pages 31–35, 2003.

[3] *The Network Simulator - NS2*. `http://www.isi.edu/nsnam/ns`.

[4] J. Liu, D.M. Nicol, B.J. Premore, and A.L.Poplawski. *Performance Prediction of a Parallel Simulator*. In Proc. of the Parallel and Distributed Simulation Conference (PADS'99), pages 156–164, 1999.

[5] A. Varga. *OMNeT++*. IEEE Network Interactive (online), 16(4), 2002.

[6] Atsuko Takefusa, Satoshi Matsuoka, Henri Casanova, and Francine Berman. *A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid*. In HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), pages 406–415, 2001.

[7] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, Kenjiro Taura, and Andrew A. Chien. *The MicroGrid: a Scientific Tool for Modeling Computational Grids*. In Proc. of Supercomputing '00 - on CD-ROM, 2000.

[8] Arnaud Legrand, Loris Marchal, and Henri Casanova. *Scheduling Distributed Applications: the SimGrid Simulation Framework*. In CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid, pages 138–145, 2003.

[9] R. Buyya and M. Murshed. *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*. The Journal of Concurrency and Computation: Practice and Experience (CCPE), 14:1175–1220, May 2002.

[10] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. *Theory and Practice in Parallel Job Scheduling*. In Dror G. Feitelson and Larry Rudolph, editors, Job Scheduling Strategies for Parallel Processing, pages 1–34. Springer Verlag, 1997.

[11] L. Hall, A. Schulz, D. Shmoys, and J. Wein. *Scheduling To Minimize Average Completion Time: Off-line and On-line Algorithms*. In SODA: ACM-SIAM Symposium on Discrete Algorithms (Conference on Theoretical and Experimental Analysis of Discrete Algorithms), pages 142–151, 1996.

[12] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. *Enhanced Algorithms for Multi-Site Scheduling*. In 3rd International Workshop on Grid Computing (Grid2002), pages 219–231, 2002.

[13] A.I.D. Bucur and D.H.J. Epema. *An Evaluation of Processor Co-Allocation for Different System Configurations and Job Structures*. In Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing, pages 195–203, 2002.

[14] A.I.D. Bucur and D.H.J. Epema. *The Influence of the Structure and Sizes of Jobs on the Performance of Co-Allocation*. In Proceedings of the Sixth Workshop on Job Scheduling Strategies for Parallel Processing, pages 154–173, 2000.

[15] K. Ranganathan and I. Foster. *Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids*. Journal of Grid Computing, 1:53–62, 2003.

[16] *The DataGrid Project*. `http://eu-datagrid.web.cern.ch/eu-datagrid/`.

[17] David G. Cameron, Ruben Carvajal-Schiaffino, A. Paul Millar, Caitriana Nicholson, Kurt Stockinger, and Floriano Zini. *Evaluating Scheduling and Replica Optimisation Strategies in OptorSim*. In 4th International Workshop on Grid Computing (Grid2003), pages 52–59, 2003.

[18] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. *SETI@home: An Experiment in Public-Resource Computing*. Communications of the ACM, 45:56–61, 2002.

[19] H. Casanova, T. Bartol, J. Stiles, and F. Berman. *Distributing MCell Simulations on the Grid*. The International Journal of High Performance Computing Applications, pages 243–257, 2001.

[20] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. *Bandwidth-centric allocation of independent tasks on heterogeneous platforms*. Technical Report 4210, INRIA, Rhone-Alpes, 2001.

[21] F. Berman et al. *Adaptive Computing on the Grid Using AppLeS*. IEEE Transactions on Parallel and Distributed Systems, 14:369–382, 2003.

[22] M. Hovestadt, O. Kao, A. Keller, and A. Streit. *Scheduling in HPC Resource Management Systems: Queueing vs. Planning*. In Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2862/2003, pages 1–20, 2003.

[23] D. Lu and P. Dinda. *GridG: Generating Realistic Computational Grids*. ACM SIGMETRICS Performance Evaluation Review, 40(4), 2003.

Algorithm B.4.1: NETWORK AWARE-PREFERLOCAL($Jobs$)

for each $j \in Jobs$
 $LocalCR \leftarrow getCR(getHome(j))$
 $LocalSR \leftarrow getSR(getHome(j))$
 $LocalIR \leftarrow getIR(getHome(j))$
 $Conn \leftarrow getConn(LocalCR, LocalSR, LocalIR)$
 $LocalResources \leftarrow [LocalCR, LocalSR, LocalIR, Conn]$
 if $canSchedule(j, LocalResources)$
  $Time \leftarrow getTimeSpan(j, LocalResources)$
  $schedule(j, LocalResources, Time)$
  $updateResourceLoads(j, LocalResources)$
 else
  $BestResources \leftarrow [\,]$
  $LeastTime \leftarrow +\infty$
  for each $c \in CRs(j), s \in SRs(j), i \in IRs(j)$
   $Conn \leftarrow getConn(c, s, i)$
   $Resources \leftarrow [c, s, i, Conn]$
   if $canSchedule(j, Resources)$
    $Time \leftarrow getTimeSpan(j, Resources)$
    if $Time < LeastTime$
     $LeastTime \leftarrow Time$
     $BestResources \leftarrow Resources$
    endif
   endif
  endfor
  if $BestResources \neq [\,]$
   $schedule(j, BestResources, LeastTime)$
   $updateResourceLoads(j, BestResources)$
  endif
 endif
endfor

C

Flexible Grid Service Management through Resource Partitioning

B. Volckaert, P. Thysebaert, M. De Leenheer, F. De Turck, B. Dhoedt, P. Demeester

accepted for publication in the Journal of Supercomputing, 2006

Abstract *In this paper, a distributed and scalable Grid service management architecture is presented. The proposed architecture is capable of monitoring task submission behavior and deriving Grid service class characteristics, for use in performing automated computational, storage and network resource-to-service partitioning. This partitioning of Grid resources amongst service classes (each service class is assigned exclusive usage of a distinct subset of the available Grid resources), along with the dynamic deployment of Grid management components dedicated and tuned to the requirements of a particular service class introduces the concept of Virtual Private Grids. We present two distinct algorithmic approaches for the resource partitioning problem, the first based on Divisible Load Theory (DLT) and the second built on Genetic Algorithms (GA). The advantages and drawbacks of each approach are discussed and their performance is evaluated on a sample Grid topology using NSGrid, an ns-2 based Grid simulator. Results show that the use of this Service Management Architecture in combination with the proposed algorithms improves computational and network resource efficiency, simplifies schedule making decisions, reduces the overall complexity of managing*

the Grid system, and at the same time improves Grid QoS support (with regard to job response times) by automatically assigning Grid resources to the different service classes prior to scheduling.

C.1 Introduction

As more and more application types are ported to Grid environments, an evolution is noticed from purely computational and/or data Grid offerings to full-scale service Grids [1] (e.g. the EGEE Enabling Grids for E-Science in Europe project [2]). In this paper, a 'service Grid' denotes a Grid infrastructure capable of supporting a multitude of *application types* with varying QoS levels (i.e. our definition of Service Grid is not limited to web-service enabled Grids). We use the term 'service class' as a classifier for user-submitted Grid jobs that exhibit similar resource requirements (processing requirements, I/O data requirements, priority, etc.). The architectural standards for Service Grids are provided by the Global Grid Forum's Open Grid Service Architecture (OGSA) [3], and (to a lesser extent) the Web Service Resource Framework [4], building on concepts of both Grid and Web Service communities.

Widespread Grid adoption also increases the need for automated distributed management of Grids, as the number of resources offered on these Grids rises dramatically (hence the scalability of these Grids becomes very important). Automated self-configuration and self-optimization of Grid resource usage can greatly reduce the cost of managing a large-scale Grid system, and at the same time achieve better resource efficiency, scalability and QoS support [5, 6].

The distributed service management architecture proposed in this paper can be described as a distinct implementation of the OGSA 'Service Level Manager' concept. Service Level Managers are, according to the OGSA specification, responsible for setting and adjusting policies, and changing the behavior of managed resources in response to observed conditions.

Our main goal is to automatically and intelligently assign Grid resources (both network, computing and data/storage resources) to a particular service class for exclusive use during a specified time frame (i.e. partitioning the pool of Grid resources into distinct service class-assigned resource pool subsets). The decision to assign a resource to one particular service will be based on the resources available to the Grid and monitored service class resource usage characteristics and requirements. Once resource partitioning has been performed, dedicated management components (i.e. scheduler, information service, etc.) will be associated to a service class's assigned resources, effectively constructing multiple self-managing 'Virtual Private Grids'. These Virtual Private Grids in turn improve Grid management scalability, as their management components only need to take into account the state of their partition-assigned resources along with the state and requirements

of jobs from the service class they are responsible for.

In order to compare the performance of a service managed Grid with a non-service managed Grid we use NSGrid (for a detailed discussion see [7]), an ns-2 based Grid simulator capable of accurately modeling different Grid resources, management components and network interconnections. More specifically, we evaluated Grid performance (in terms of average job response time and resource usage efficiency) when different partitioning strategies are employed, and this both in case network aware as when network unaware scheduling is used.

This paper is structured as follows: section C.2 summarizes related work in this area, while section C.3 provides details on the proposed service management architecture and its interaction with other Grid components. The employed network and non network aware scheduling algorithms are highlighted in section C.4. Section C.5 elaborates on the different resource partitioning strategies, while the evaluation of those partitioning strategies in a typical Grid topology is compared to a non-resource partitioned situation for varying job loads in section C.6. Finally, section C.7 presents some concluding remarks.

C.2 Related Work

Considerable work has already been done in the area of distributed scheduling for Grids [8]. Grid scheduling taking into account service specific requirements has been dubbed *application-level scheduling*. Most notable application-level research projects include AppLeS [9] and GrADS [10].

In AppLeS, service-class scheduling agents inter-operable with existing resource management systems have been implemented. Essentially, one separate scheduler needs to be constructed per application type. Our service management architecture differs from this approach in that it operates completely separated from the Grid scheduling components, working in on service-exclusivity properties located at the Information Services (responsible for storing resource properties and answering resource queries from e.g. the different schedulers).

GrADS on the other hand is a project to provide an end-to-end Grid application preparation and execution environment. Application run-time specific resource information comes from the Network Weather Service [11] and MDS2 [12]. For each application; a performance (i.e. computational, memory and communication) model needs to be provided by the user. This differs from our Service Monitor approach, which actively monitors application behavior and deduces service characteristics at run-time (see section C.3.3).

The General purpose Architecture for Reservation and Allocation (GARA) project [13] provides Globus with end-to-end Quality of Service guarantees for applications. Both advance and immediate resource reservations are supported. GARA does not offer dynamic automated resource-to-service partitioning but can

instead be seen as a technology enabling the work proposed in this paper.

IBM's Tivoli Intelligent Orchestrator (TIO) and Provisioning Manager (TPM) [14] can improve service response times by monitoring registered resources and requirements for anticipated peak workloads and, if necessary, can automatically re-allocate resources in accordance with business priorities. TIO and TPM are focused on automated data center resource-to-service allocations, and require users to predefine 'optimal resource utilization' plans for each supported service class. Our service management architecture focuses on the needs of generic computational / data / service Grids, and tries to automatically (i.e. without user interaction) deduce optimal resource utilization from monitored Grid job submission behaviour.

Optimally assigning resources to services has been the subject of research in [15]. In this study however, resource selection occurs each time a job is submitted to a Grid Portal (i.e. service aware scheduling). This differs from the work proposed in this paper in which resources are pre-assigned to service classes based on service class characteristics (i.e. prior to the job scheduling process).

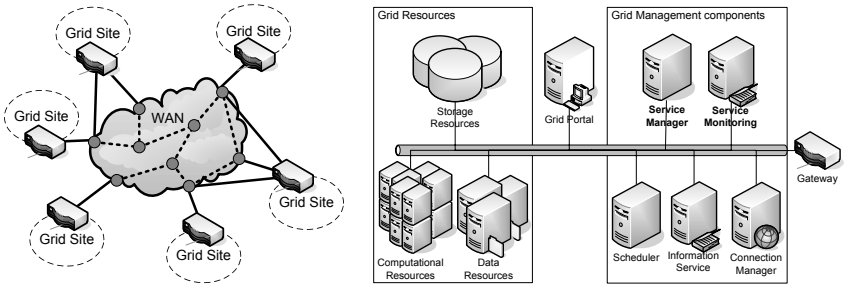
In contrast to the above mentioned research projects, our contribution focuses on distributed, automated and intelligent resource-to-service partitioning in a Grid environment (based on monitored service class characteristics/requirements) along with the dynamic deployment of service class exclusive management components (effectively constructing multiple Virtual Private Grids).

C.3 Service Management Concept

In this section we begin by describing the NSGrid models that are employed: Grid Site (resources, management components, etc.) and job models are discussed, along with basic job submission / resource assignment protocols. We continue by discussing the overall concept of resource-to-service partitioning in subsection C.3.2 and explain in subsection C.3.3 how our resource-to-service partitioning architecture was implemented in NSGrid.

C.3.1 Grid/Job Model

We regard a Grid as a collection of *Grid Sites* interconnected by WAN links (see figure C.1). Each Grid Site has its own resources (computational, storage and data resources) and a set of management components, all of which are interconnected by means of LAN links. Management components include a *Connection Manager* (capable of offering network QoS by providing bandwidth reservation support, and responsible for monitoring available link bandwidth and delay), an *Information Service* (storing registered resources' properties and monitoring their status) and a *Scheduler*. Every Grid resource in our model is given an associated service



(a) Grid (b) Grid Site

Figure C.1: Grid Model

class ID property (stored in the Information Service with which the resource is registered). If no Service Management components are instantiated in the Grid, all resources' service class ID equals '0', meaning these resources can be used by *any* job (i.e. belonging to *any* service class).

The basic unit of work in our model is a *job*, which can roughly be characterized by its length (time it takes to execute on a reference processor), computational requirements (memory, operating system, installed applications, etc.), the needed input data, the output data size, the *burstiness* with which these data streams are read or written, and the service class to which it belongs. A job's service class ID can either be assigned by the Grid application from which this job was spawned (with a unique service class ID per Grid application), or alternatively jobs from different applications but with similar monitored resource requirements can be given the same service class ID by the service monitor (the latter approach is useful if one or more Grid applications spawn jobs with widely differing requirements/characteristics rendering application-based service class ID assignments less interesting). Knowing the job's total length and the frequency at which each input (output) stream is read (written), the total execution length of a job can be seen as a concatenation of instruction "blocks". The block of input data to be processed in such an instruction block is to be present before the start of the instruction block; that data is therefore transferred from the input source at the start of the previous instruction block. Similarly, the output data produced by each instruction block is sent out at the beginning of the next instruction block. We assume these input and output transfers occur in parallel with the execution of an instruction block. Only when input data is not available at the beginning of an instruction block or previous output data has not been completely transferred yet, a job is suspended until the blocking operation completes. A typical job execution cycle (one input stream and one output stream) is shown in figure C.2. The presented model allows us to mimic both *streaming* data (high read or write fre-

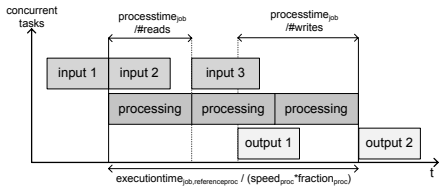


Figure C.2: Non-blocking job, simultaneous transfer and execution

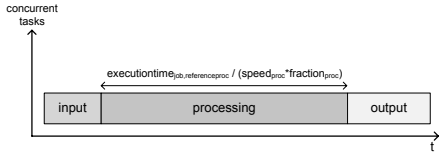


Figure C.3: Non-blocking job, pre-staged input data

quency) and *data staging* approaches (number of input/output blocks set to 1 as can be seen in C.3).

In NSGrid [7], when a simulated client submits jobs, the exact job properties are generated from pre-configured job distributions. Each Grid Site has one or more *Grid Portals* through which clients can submit their jobs. Once submitted, a job gets queued at the local *Scheduler*, which in turn queries the Information Services (IS) (located both at the local site and at foreign sites) for resources adhering to the job's requirements. Once the next scheduling round starts, the Scheduler applies one of its scheduling algorithms and (if possible) selects one or more data resources (DR) (for job input data), together with one or more storage resources (SR) (for storing job output data) and a computational resource (CR) (providing job processing power), all *not necessarily* located at one Grid Site (note that DRs and SRs can reside on the same network node - modeling one data/storage capable resource).

If the scheduling algorithm is network aware (see figure C.8), the Connection Manager (CM) is queried for information about available bandwidth on (shortest route) paths between resources and, once a scheduling decision is made (taking into account the speed at which I/O data can be fetched/stored to/from the processing job and adjusting computational power that gets reserved for this job to match), attempts to make connection reservations between the selected resources; connection reservations provide a guaranteed minimum bandwidth available for that job. Note that reservations are not physically set up by the Connection Manager: if the bandwidth requirements of the requested connection reservation are not infringing previously guaranteed connection reservations' minimum bandwidth, the request is granted. If however this is not the case (due to the use of stale resource state information when assigning resources to jobs in the scheduling round), the con-

nection reservation request is rejected and the job will be put back in the scheduler queue until the next scheduling round. The Connection Manager thus operates by bookkeeping all granted connection reservations and denying new reservations that would infringe on those previously granted reservations. Once all resource reservations are successful, the job is sent to the selected computational resource which takes care of fetching the different input datasets and storing the job's output data.

C.3.2 Resource Partitioning

Our goal is to intelligently and automatically assign service class IDs to each resource so they can be used exclusively for jobs spawned from that service class. This classification of Grid resources in a per-service resource pool with its own dedicated scheduler and information service has multiple benefits:

- resource efficiency and average job response times improve (as will be shown in section C.6)
- allows for faster scheduling decisions and resource information lookups
- service class priorities can be given by assigning more resources to high-priority service classes
- locally offered service classes can be prioritized over foreign Grid Site service classes,
- reduced infrastructure costs: by allocating job loads to resources more efficiently, the number of resources can be reduced,
- improved scalability with dynamic deployment of dedicated VPG management components,
- service class dedicated management components can be fine-tuned to the needs of their particular service.

As we will see in section C.6, resource efficiency (and average job response times) can be improved by limiting resource availability to service classes that can make efficient use of that particular resource (e.g. taking into account service class' data locality). In addition, the number of job resource query results returned by the Information Services to the scheduler will be less than when there is one common resource pool, allowing for faster scheduling decisions (as we are in fact utilizing the resources' service class ID assignation as an advance reservation mechanism).

Of course, one has to be very careful when automatically assigning resources to service classes, as it creates the risk that certain service classes are (involuntarily) left starving for resources on which to run, while other resources are assigned to

a service class for which there are no job submissions at that time (and are thus unnecessarily left idle). One also has to take into account service class necessities when making resource partitioning decisions, in order to avoid excluding a service class from access to a critical resource (e.g. prohibiting a service class access to mandatory data resources).

The same way computational, storage and data resources can be partitioned amongst different service class resource pools, network resources can also be split up by performing per-service bandwidth reservations (e.g. VPN technology). This can prevent data-intensive service classes from monopolizing network bandwidth usage and thereby hampering the performance of jobs from other service classes (see figure C.4). Instead, each service class should automatically receive a certain bandwidth and be able to use this bandwidth without having to worry about the network usage of other services' jobs.

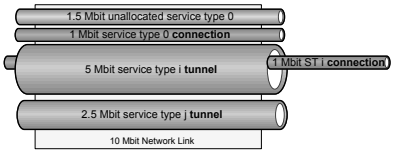
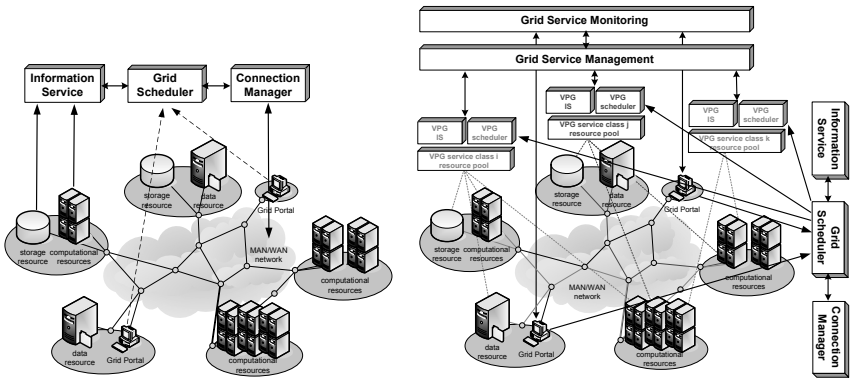


Figure C.4: Network Resource Partitioning

With combined network and resource partitioning, a Grid can be modeled as a dynamic collection of overlay Grids or *Virtual Private Grids* (VPG), with one VPG for each service class offered in the Grid. These VPGs (see figure C.5) are not static structures in that they do not have resources assigned to them in a permanent way, but react to monitored changes in service characteristics (e.g. additional service offerings can lead to the construction of new VPGs and reallocation of resources across existing VPGs). Resource reallocation can stem from important changes in monitored service class characteristics (e.g. higher job submission rates for a service class), a change in service class priorities or, as already mentioned, the addition of new service classes.

C.3.3 NSGrid implementation

In NSGrid, a distributed service management architecture was implemented in order to evaluate the effectiveness of different resource-to-service partitioning strategies and Virtual Private Grid deployments. Each Grid Site typically has a local *Service Manager*, which interacts with the local Information Service (IS), Connection Manager (CM) and *Service Monitor* (see figure C.7 for a sample Service Management setup in NSGrid). All NSGrid resources and management components are located at ns-2 nodes, which can be interconnected by means of different types of network links with configurable bandwidth and delay. This way, all job



(a) standard Grid (b) VPG partitioned Grid

Figure C.5: Standard Grid architecture vs. Virtual Private Grid partitioned Grid architecture

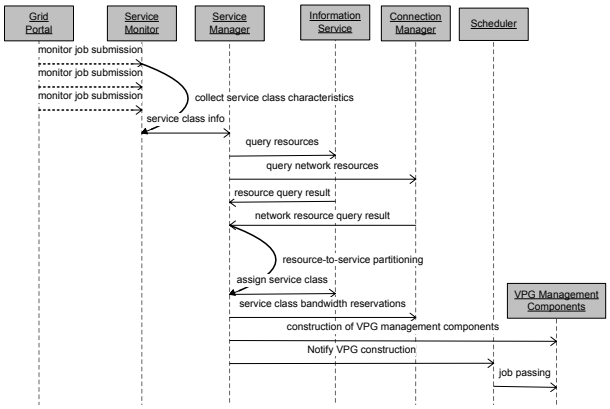


Figure C.6: VPG partitioning messages

I/O data that is sent between the different resources is accurately simulated by ns-2, allowing us to monitor bandwidth usage, network congestion, etc.. All control messages are XML-encoded and sent over the underlying ns-2 network.

C.3.3.1 Service Monitor

The Service Monitor inspects job submission behavior at the Grid portals (recall that a Grid portal acts as a job submission gateway for Grid users): each time a job is submitted, job requirements (service class, priority, needed input data sets and sizes, output storage sizes, computational requirements, etc.) are extracted and overall service class properties (e.g. average job interarrival time, average I/O

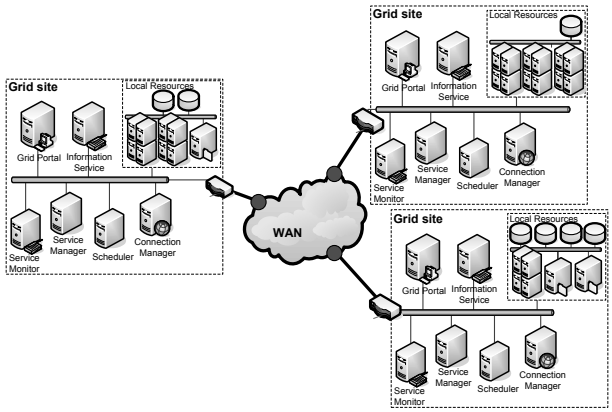


Figure C.7: NSGrid Service Management Architecture scenario

data sizes, average job computational needs, needed input datasets) are adjusted. When the Service Monitor has gathered adequate service class characteristics (either when service class properties remain relatively stable over a fixed period of time, or when an information dissemination timer has run out), the Service Monitor sends the collected service class' characteristics to its known (local and foreign Grid site) Service Managers, so as to allow them to have up-to-date service class information for use by the resource-to-service partitioning algorithms. The Service Monitor keeps a record of the info that was submitted to the Service Managers, and, if substantial changes (w.r.t. a configurable threshold) in service class properties are monitored (e.g. detection of new service classes, increased service class job interarrival times, change in priority, higher job response times, etc.), sends up-to-date service class information to the Service Managers (see figure C.6).

Note that when a job from a newly monitored service class is detected *and* all resources have been assigned to existing service classes (i.e. non service class 0 assignations), the job will have to wait for a repartitioning of resources before being able to be scheduled. The Service Monitor will wait until the newly monitored average service class characteristics have stabilized, or until its information dissemination timer has run out, after which it will contact the Service Manager informing it of the existence and characteristics of the newly monitored service class.

Each Service Monitor has a moving time window (of configurable length), such that the properties of a job that was submitted at a time before the time window's beginning are no longer taken into account when calculating service class' characteristics. In doing so, service classes that spawn no jobs during a period of time equal to the time window's length are discarded: the Service Monitor will inform the Service Manager of this occurrence, which in turn will free resources

allocated to that particular service class and (if necessary) repartition.

C.3.3.2 Service Manager

The Service Manager thus periodically receives information regarding local and foreign Grid site service class characteristics from the different Service Monitors. When the received information does not differ (with regard to a certain threshold) from the one used to partition the Grid resources in a previous partitioning run, no resource-to-service repartitioning will occur. If however the difference between the previous values and currently monitored service characteristics (average job IAT, processing length, I/O bandwidth necessities, etc.) is too large, or if no resource partitioning has yet been done, the Service Manager will query the Information Services for the characteristics of the resources in their local Grid site resource pool. Once the answer to this query has been received, one of the resource partitioning algorithms (detailed in section C.5) is applied to the resource set, and the resulting resource partitioning solution is sent back to the Information Services, who in turn change the service class property of their registered resources. If the partitioning algorithm also works in on network resources, the Connection Manager will be contacted to make service bandwidth reservations (based on assigned computational resources, necessary input datasets and monitored service class' bandwidth requirements).

Once the partitioning algorithm has finished, resources will be assigned to service class resource pools, and (if this was not already done) dedicated Virtual Private Grid management components will be dynamically constructed and associated with the different Virtual Private Grids (in NSGrid these VPG management components are deployed at the Grid site where jobs from the VPG's service class are most common). A VPG Information Service will gather resource property and status information from all resources assigned to the VPG. This Information Service will in turn be queried by a dedicated VPG scheduler when the latter seeks information on resources adhering to a job's requirements. Note that the global (central or distributed) Grid scheduling system continues to receive all jobs submitted to the different Grid portals, but, upon inspection of the service class of each arriving job, either tries to schedule the job itself, or, when a VPG is constructed for the job's service class, immediately sends it to the dedicated VPG scheduler.

C.3.3.3 Information Service

Much in the same way as the Service Monitors can trigger a repartitioning of resources to services when substantial changes in service class characteristics are monitored, the Information Services are responsible for signaling changes in resource availability. Every time an existing Grid resource becomes unavailable (either because of failure or by policy), or conversely, when new resources become

available to the Grid, the Information Services report this change to the Service Manager. The latter then decides if a resource-to-service repartitioning is necessary.

It is important to note that, while resources are assigned for exclusive use by a particular service, not one job using a service class reassigned resource will be interrupted (preventing jobs from being pre-empted when the CR it is running on is assigned to a different service class). The service assignment will thus only be effective for *new* jobs or jobs currently in the scheduler queue. At the time of scheduling, queries will be sent to the Information Services for resources adhering to the job's requirements, and these Information Services will return only those resources that are assigned to that particular job's service class.

C.4 Scheduling Strategies

When jobs are submitted, a Scheduler needs to decide where to place the job for execution. The scheduling algorithm used in making this selection has a big impact on Grid performance, and influences overall Grid job throughput, Grid resource efficiency etc. All presented algorithms are *queueing* algorithms [16], that is, whenever an algorithm is invoked, it will attempt to schedule the not-yet scheduled jobs in the order of arrival on the time-shared resources. Jobs that cannot be scheduled will be requeued, preserving the relative order of arrival (note that other requeueing methods are available). The time between two scheduling rounds can be fixed, but it is also possible to set a threshold (e.g. time limit or number of unscheduled jobs in the queue) which triggers the next scheduling round. As the goal of each algorithm is the minimization of each job's response time, a natural metric to benchmark the different algorithms is the average job *turnaround time*. In what follows we will briefly explain the different scheduling strategies used in our simulations (for a more detailed discussion see [17]). Once scheduled, our scheduler does not attempt to pre-empt jobs.

C.4.1 Non-Network Aware Scheduling

Non-Network aware scheduling will compute Grid job schedules based on the status of the computational, storage and data resources (as provided by the Information Services). Algorithms that use this kind of approach will not take into account information concerning the status of resource interconnections. The decision of which resources to use for a job will be based on the information acquired from the different Information Services (i.e. job execution speed and end time will be calculated based on the status of CR/SR/DR). It is precisely because Non-Network aware algorithms assume that residual bandwidth on network links is "sufficient", that jobs can block on I/O operations: their computational progress

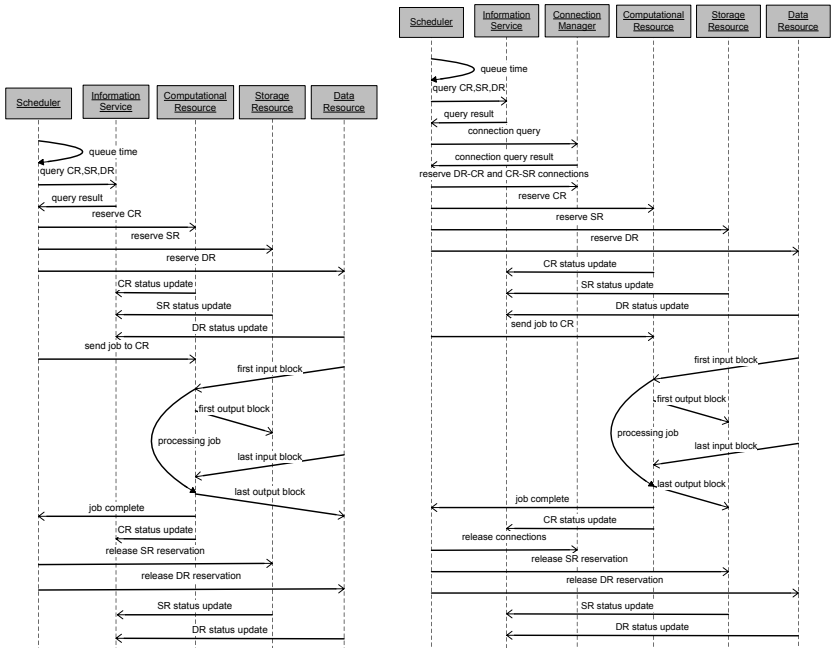
is no longer only determined by the computational resource's processor fraction that has been allocated to it (which, together with the job's length and the computational resource's relative speed determines its earliest end time *if all input and output transfers complete on time i.e. before the start of the appropriate instruction block*), but also by the limited bandwidth available to its input and output streams. Note that the fact that network information is discarded during the scheduling implies that no connection reservations (providing guaranteed available bandwidths) are made with the connection manager - these would allow to accurately predict the job's running time.

C.4.2 Network Aware Scheduling

Network aware scheduling algorithms will not only contact the Information Services (for information about resources that adhere to the job's requirements), but will also query the Connection Manager for information about the status of the network links interconnecting these resources (i.e. the Connection Manager will send the Grid Scheduler information about connections that can be set up between DR/CR couples (necessary for job input retrieval) and CR/SR couples (needed for job output storing)). Based on the answers from the Information Services and Connection Manager, the scheduling algorithm is able to calculate job execution speed and end time more accurately, taking into account the speed at which input/output can be delivered to each available computational resource. For jobs with 1 input stream and 1 output stream, the best resource (CR/SR/DR) triplet is the one that minimizes the expected completion time of the job. This value is determined by the available processing power to that job on the computational resource (and its relative speed), the job's length, the job's total input and output data size and the residual bandwidth on the observed links from DR to CR and from CR to SR.

As explained, for some (CR,SR,DR) triplet, due to bandwidth constraints, this duration may be significantly higher than the value calculated from the job's length and the CR's relative speed, even if job execution and data transfer occur simultaneously. The scheduler selects the optimal CR/DR/SR triplet and contacts the central Connection Manager to perform the necessary connection setups (the necessary bandwidth of these connections is calculated by the scheduler). The job then gets transferred to the selected CR for processing and input/output is sent from/to the DR/SR over the reserved connections. If no (local or remote) resources satisfying the job's requirements can be found, or if no connections with sufficient bandwidth are available, the job will be queued and prepared for reschedulement. The time it takes for a job to complete since it has been submitted by the client can be broken up into:

- sending the job to the scheduler
- time spent in the scheduler's queue



(a) Non-network aware job scheduling (b) Network aware job scheduling

Figure C.8: NSGrid non-network aware versus network aware scheduling

- time needed for the co-allocation of resources (including network resources) allocated to that job
- transfer time for the first input data block(s)
- time needed to process the job at its allocated execution speed
- transfer time for the last output data block(s).

Each of these can be found in figure C.8. Note that no job can become blocked because reservations are made with network resources, excluding the network from becoming an unexpected bottleneck (if the resource state information returned by the Information Services/Connection Managers and employed by the scheduler was accurate and up-to-date).

C.5 Partitioning Strategies

The problem at hand is trying to partition resources into service class resource pools. A solution in this case is a mapping from resource to a particular service

class ID, and this for all resources returned from the Service Manager - Information Service queries. A resource can also be assigned service class ID '0', meaning it can be used by jobs from every service class. Exhaustively searching for an optimal partitioning (by evaluating the fitness of a solution by means of a cost function) quickly becomes infeasible, as the amount of solutions that needs to be evaluated is $(\#serviceclasses + 1)^{\#resources}$. In our attempts to find a suitable solution in reasonable time, we have used two distinct approaches: one uses Divisible Load Theory (DLT) to obtain a tractable Integer Linear Program (ILP) modeling the service class assignment problem, while the other uses a Genetic Algorithm to obtain a resource-to-service mapping.

C.5.1 DLT based Partitioning

Whenever a Grid reaches a steady state (e.g. a Grid processing a periodic load), stochastic parameters regarding the distributions of job interarrival time, duration and I/O-needs can be estimated for each service class by the Service Monitoring Architecture. These parameters can then be used to populate an Integer Linear Program designed to (by assigning appropriate values for the program's decision variables)

1. Assign an exclusive service class ID to each computational resource
2. Determine the optimal *schedule* of the periodic workload over the Grid's resources, taking into account the resource-to-service assignation

An approximation used to limit the number of integer variables in this approach is to treat the aggregate workload as arbitrarily divisible (hence the name "Divisible Load Theory") [18, 19]. In this context, values of interest are $arrivals_s^n$ - the computational load per time unit arriving at site s and belonging to service class n, $Sets_n$ and $Size_n$ - the datasets available to service class n jobs and their respective sizes. The main decision variables in the problem are $x_{c,n}$ (binary, assigning service class n exclusive access to CR c) and $\alpha_{i,n}^c$ (real-valued, amount of service class n computational load per time unit processed at CR c which arrived at site i). Auxiliary variables needed to fulfill routing constraints on the input datasets and generated output data have been dubbed $in_{n,j}^l$ (bandwidth needed on link l for transport of dataset j of service class n) and out_s^l (bandwidth needed on link l for transport of output data to storage resource s) - note that the concept of source-based routing [20] was used to formulate the routing constraints.

Using the Divisible Load approach, the resource-to-service assignation can now be modeled as a cost minimization problem with several classes of constraints[1].

[1]Abbreviations used: GW = Gateways, L^+ = outgoing links, L^- = incoming links, Cap_c=computational res. capacity, Cap_l=link capacity

The capacity constraints to be observed for each computational Resource and Network Link, respectively, are

$$\forall c \in CR. \sum_{i \in Sites} \sum_{n \in SC} \alpha_{i,n}^{c} \leq Cap_c \tag{C.1}$$

$$\forall l \in L. \sum_{n \in SC} \sum_{j \in Sets_n} in_{n,j}^{l} + \sum_{s \in SR} out_{s}^{l} \leq Cap_l \tag{C.2}$$

These constraints ensure that work allocated to a Computational Resource does not exceed that resource's processing capacity, and that total network traffic over each link does not exceed that link's capacity. Network traffic is routed according to following constraints:

$$\forall n \in SC, j \in Sets_n. \sum_{d \in DR: j \in Sets_d} \sum_{l \in L_d^+} in_{n,j}^{l} = \frac{\sum_{s \in Sites} arrivals_s^n \times Size_n}{\#Sets_n} \tag{C.3}$$

$$\forall c \in CR, n \in SC, j \in Sets_n. \sum_{l \in L_c^-} in_{n,j}^{l} = \frac{\sum_{i \in Sites} \alpha_{i,n}^{c} \times Size_n}{\#Sets_n} \tag{C.4}$$

$$\forall c \in CR, s \in SR. \sum_{l \in L_c^+} out_{s}^{l} = \sum_{n \in SC} \alpha_{Site_s,n}^{c} \times Size_n \tag{C.5}$$

$$\forall s \in SR. \sum_{l \in L_s^-} out_{s}^{l} = \sum_{n \in SC} arrivals_{Site_s}^{n} \times Size_n \tag{C.6}$$

$$\forall g \in GW, n \in SC, j \in Sets_n. \sum_{l \in L_g^-} in_{n,j}^{l} = \sum_{l \in L_g^+} in_{n,j}^{l} \tag{C.7}$$

$$\forall g \in GW, s \in SR. \sum_{l \in L_g^-} out_{s}^{l} = \sum_{l \in L_g^+} out_{s}^{l} \tag{C.8}$$

The first two equations in this series describe how much traffic is carried on the network links departing from the Data Resources, given that any job of a given service class has an equal probability to process any of the data sets available to that service class. That same amount of network traffic is of course to be retrieved at the Computational Resource side.

The next two equations present the analogous observation for output data generated by the jobs.

The last two equations state that network flow (both for input and output data) is conserved when crossing intermediate routers.

A feasible schedule is obtained by demanding that the total distributed workload equals the size of the arriving workload per time unit:

$$\forall i \in sites, n \in SC. \sum_{c \in CR} \alpha_{i,n}^{c} = arrivals_i^n \tag{C.9}$$

Constraints concerning the exclusive reservation of each CR:

$$\forall c \in CR. \sum_{n \in SC} x_{c,n} = 1 \tag{C.10}$$

$$\forall c \in CR, n \in SC. \sum_{i \in Sites} \alpha_{i,n}^{c} \leq x_{c,n} \times Cap_c \tag{C.11}$$

where the last equation is used to express that only those Computational Resources which have been explicitly assigned to a service class may actually perform work in that service class.

The "cost" to be minimized can take on several forms; for instance, the total amount of data traveling over network links per unit of time (in the steady-state Grid) can be described in terms of problem variables as

$$\sum_{l \in L} \left(\sum_{n \in SC, j \in Sets_n} in_{n,j}^{l} + \sum_{s \in SR} out_{s}^{l} \right) \tag{C.12}$$

Using this cost function in the ILP results in a workload schedule and service class assignation yielding minimal aggregate network load for a given arrival process. Alternatively, one can choose to minimize the maximal unused computational resource fraction, which results in an "even" workload distribution across all computational resources according to their respective capacities. This approach can be modeled by adding the constraints

$$\forall c \in CR, n \in SC.cost \geq \frac{\left(x_{c,n} \times Cap_c - \sum_{i \in Sites} \alpha_{i,n}^{c}\right)}{Cap_c} \tag{C.13}$$

and minimizing the cost.

C.5.2 Genetic Algorithm Heuristic

The resource class assignment can easily be encoded into an n-tuple of service class IDs, where n equals the number of resources. These *chromosomes* can then be fed to a Genetic Algorithm (GA) which evaluates the fitness of each chromosome (i.e. possible service class assignment) w.r.t. a cost function $f(x)$ (see algorithm C.5.1). Unlike with an Integer Linear Program, this cost function needs not be "linear" in the decision variables, giving this partitioning approach more expressive power than the DLT based partitioning.

Algorithm C.5.1 starts with an initial population size of m randomly generated tuples (each tuple b consisting of n service class ID slots). While the stop condition is not fulfilled, the GA applies a proportional selection, after which a two-point crossover and a mutation step occur. The proportional selection selects tuples based on their fitness (with fitter solutions more likely to be selected and

carried over to the next generation). In the next step, a two-point crossover operation is applied (for each two consecutive tuples the crossover probability ρ_C determines if all service class IDs between the randomly selected $pos1$ and $pos2$ are switched). Finally, the mutation operation is performed for each tuple, with mutation probability ρ_M determining which of the n service class ID slots needs to be mutated to a random service class ID.

Depending on how much time is available between partitioning runs (which in turn depends on the stability of the different service characteristics), parameters of this GA can be tuned in such a way that feasible search times can be attained (i.e. search time $<<$ time between partitioning runs).

In the next sections we provide details on some implemented partitioning strategies (and accompanying cost functions): section C.5.2.1 and section C.5.2.2 describe computational resource partitioning based on the processing requirements of respectively local and global service classes. Taking into account the site locality of much needed service class' input datasets is discussed in section C.5.2.3. Finally, partitioning of network resources based on data requirements of the different service classes is discussed in section C.5.2.4. We assume that the Service Manager has received both up-to-date local and foreign Grid Site service characteristics from the Service Monitors and resource properties from the Information Services.

C.5.2.1 Local Service CR Partitioning

The first (and simplest) partitioning strategy only takes into account the computational processing needs and priority of the different *local* service classes. The Service Manager queries the Information Services for all local computational resources and calculates average service class' requested processing power as the average processing time of that service class (as measured on a CR running at reference speed) divided by the average interarrival time of that SC (the higher job interarrival times, the less processing power will be needed) and multiplied with the number of sites that submit jobs from this SC. [2]:

$$\forall SC \cdot ppower_{req_{SC}} = sites_{SC} \times \frac{ptime_{ref_{SC}}}{IAT_{SC}}$$

The relative processing power assigned to a service class (sum of processing power of computational resources assigned to that SC) can be found from[3]:

$$\forall SC \cdot ppower_{asg_{SC}} = \sum_{\forall CR \in SC} \frac{speed_{CR}}{speed_{CR_{ref}}} \times ptime_{ref_{SC}}$$

[2] $ptime_{ref_{SC}}$ = average processing time of service class SC job on reference CR, $sites_{SC}$ = amount of Grid portals launching service class SC's jobs, IAT_{SC} = average service class SC's job interarrival time

[3] $speed_{CR}$ = processing speed of CR, $speed_{CR_{ref}}$ = processing speed of reference CR

Algorithm C.5.1: GENETIC ALGORITHM($resources$)

$population_{initial} \leftarrow (b_{(1,0)}, ..., b(m,0)),\ t \leftarrow 0$
while $stopcondition$ **false**
do
- **comment:** proportional selection
- **for** $i \leftarrow 1$ **to** m
  - **do**
    - $x \leftarrow rand[0,1]$
    - $k \leftarrow 1$
    - **while** $k < m$ **and** $x < \sum_{j=1}^{k} \frac{f(b_{j,t})}{\sum_{j=1}^{m} f(b_{j,t})}$
      - **do** $k \leftarrow k + 1$
    - $b_{i,t+1} \leftarrow b_{k,t}$
- **comment:** two-point crossover
- **for** $i \leftarrow 1$ **to** $m - 1$ **step** $i + 2$
  - **do**
    - **if** $rand[0,1] \leq \rho_C$
      - **then**
        - $pos1 \leftarrow rand[1,n]$
        - $pos2 \leftarrow rand[1,n]$
        - **if** $pos1 > pos2$
          - **then** $switch(pos1, pos2)$
        - **for** $k \leftarrow pos1$ **to** $pos2$
          - **do** $switch(b_{i,t+1}[k], b_{i+1,t+1}[k])$
- **comment:** mutation
- **for** $i \leftarrow 1$ **to** m
  - **do**
    - **for** $k \leftarrow 1$ **to** n
      - **do**
        - **if** $rand[0,1] < \rho_M$
          - **then** $b_{i,t+1}[k] \leftarrow rand[0, \#SC]$
- $t \leftarrow t + 1$

Once CR query answers have been received, the GA (as shown in algorithm C.5.1) will be started with cost function $f(x)$ described in algorithm C.5.2.

In this cost function (which is to be maximized), the objective is to donate to each *local* service class the same amount of processing power *relative* to their requested processing power (giving a higher cost function impact factor to service classes that have a high priority). The $maxAlloc_{over}$ and $maxAlloc_{under}$ parameters assure an even spread of processing power to services (both in case insufficient processing power is available and when sufficient processing power is available),

Algorithm C.5.2: $f_{CRpart_{local}}(x)$

$result \leftarrow \frac{ppower_{asg_0}}{2}$
$maxAlloc_{over} \leftarrow 0$
$maxAlloc_{under} \leftarrow 0$
for $i \in SC_{local}$
do {
  $aux \leftarrow ppower_{req_i} - ppower_{asg_i}$
  if $aux < 0$
    then { **if** $-aux > maxAlloc_{over}$ **then** $maxAlloc_{over} \leftarrow -aux$; $aux \leftarrow ppower_{asg_i}$
    else { **if** $\frac{aux}{ppower_{req_i}} > maxAlloc_{under}$ **then** $maxAlloc_{under} \leftarrow \frac{aux}{ppower_{req_i}}$; $aux \leftarrow ppower_{asg_i} - aux$
  $result \overset{+}{\leftarrow} \frac{priority_i}{(\sum_{j \in SC_{local}} priority_j)} \times aux$
$result \overset{-}{\leftarrow} maxAlloc_{over} + maxAlloc_{under}$
return $(result)$

as they keep track of the maximum amount of overallocated/underallocated processing power and penalize the cost function result accordingly.

C.5.2.2 Global Service CR Partitioning

The second partitioning strategy adds support for services offered at foreign grid sites. The cost function impact factor of assigning resources to foreign service classes can be adjusted by the local Service Manager by tuning the foreign service policy $\rho_{SC_{foreign}}$. Support for foreign service classes can range from no impact at all on the cost function ($\rho_{SC_{foreign}} = 0$) to an impact equal to that of local service classes ($\rho_{SC_{foreign}} = 1$) or any value in between. The resulting cost function is stated in algorithm 5.3.

C.5.2.3 Input Data Locality Penalization

Resource partitioning based solely on the processing needs of the different services can lead to bad performance. In case of data-intensive services in particular, one wants these services to be processed on computational resources located near input data that is generally requested by those service classes. In order to provide

Algorithm C.5.3: $f_{CRpart_{global}}(x)$

$result \leftarrow \frac{ppower_{asg_0}}{2}$
$maxAlloc_{over} \leftarrow 0$
$maxAlloc_{under} \leftarrow 0$
for $i \in SC_{local} \cup SC_{foreign}$
do {
  $aux \leftarrow ppower_{req_i} - ppower_{asg_i}$
  if $aux < 0$
    then {
      if $-aux > maxAlloc_{over}$
        then $maxAlloc_{over} \leftarrow -aux$
      $aux \leftarrow ppower_{asg_i}$
    }
    else {
      if $\frac{aux}{ppower_{req_i}} > maxAlloc_{under}$
        then $maxAlloc_{under} \leftarrow \frac{aux}{ppower_{req_i}}$
      $aux \leftarrow ppower_{asg_i} - aux$
    }
  if $i \in SC_{foreign}$
    then $aux \leftarrow aux \times \rho_{SC_{foreign}}$
  $result \overset{+}{\leftarrow} \frac{priority_i}{(\sum_{j \in SC} priority_j)} \times aux$
}
$result \overset{-}{\leftarrow} maxAlloc_{over} + maxAlloc_{under}$
return $(result)$

this functionality, the Service Manager queries the Information Services for both computational and data resources and constructs a list of which CRs have local access (i.e. accessible from the local Grid Site) to which input sets. We adjust the cost function to include this notion and penalize assigning a computational resource that has *no local access to an input dataset much-needed by the assigned service*. The actual penalty depends on the input data intensiveness of the service class i ($\frac{InputReq_i}{IAT_i}$) when compared to the total input data requirements of all service classes ($\sum_{\forall j \in SC} \frac{InputReq_j}{IAT_j}$): [4]

$$cost_{CR \in SCi} = \frac{\frac{InputReq_i}{IAT_i}}{\sum_{\forall j \in SC} \frac{InputReq_j}{IAT_j}} \times \frac{\rho_{cost}}{\#CR_{assigned_i}}$$

An additional (yet larger) penalty is given when, amongst all computational resources assigned to a particular service, *not one of them* has access to a needed

[4] $InputReq$ = avg. service class's input size requirement, $\#CR_{assigned}$ = amount of CRs assigned to service class, ρ_{cost} = data non-locality penalty factor

dataset, as it can be considered best practice that at least one computational resource can access a needed input set locally. This cost is only charged once for each service class.

$$cost = \frac{\frac{InputReq_i}{IAT_i}}{\sum_{\forall j \in SC} \frac{InputReq_j}{IAT_j}} \times \rho_{cost}$$

Both costs can be used as a penalty for the cost function in algorithm C.5.2 and C.5.3.

C.5.2.4 Network Partitioning

Since the Service Monitor keeps track of I/O data characteristics of each service, data intensiveness relative to the other services can be calculated. This in turn can be used to perform per-service network bandwidth reservations. We have implemented a proof-of-concept network partitioning strategy, in which the Service Manager calculates average data requirement percentages for each service class i [5]

$$bw_{req_i} = \frac{\frac{bw_{input_i} + bw_{output_i}}{IAT_i}}{\sum_{\forall j \in SC} \frac{bw_{input_j} + bw_{output_j}}{IAT_j}}$$

and passes this information to the Connection Manager, who in turn will make service class bandwidth reservations on all network links for which it is responsible. Network partitioning can be applied to all previously mentioned partitioning algorithms.

C.6 Performance Evaluation

C.6.1 Resource setup

A fixed Grid topology (see figure C.9) was used for all simulations (run on an LCG-2.6.0 Grid [21] comprised of dual Opteron 242 1.6Ghz worknodes with 2 GB RAM per CPU, and operating under Scientific Linux 3). First, a WAN topology (containing 9 core routers with an average out-degree of 3) was instantiated using the *GridG* tool [22]. Amongst the edge LANs of this topology, we have chosen 12 of them to represent a Grid site. Each site has its own resources, management components and Grid portal interconnected through 1Gbps LAN links, with Grid site interconnections consisting of dedicated 10Mbps WAN links. A single Service Manager was instantiated, and was given access to the different Grid Sites' Information Services.

[5] bw_{input} = avg. service class's input bandwidth need: $\frac{speed_{CR}}{speed_{CR_{ref}}} \times \frac{InputReq}{ptime_{ref}}$, bw_{output} = avg. service class's output bandwidth need: $\frac{speed_{CR}}{speed_{CR_{ref}}} \times \frac{OutputReq}{ptime_{ref}}$

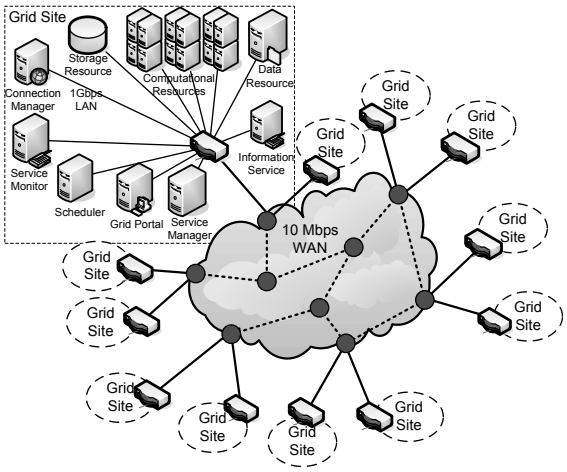


Figure C.9: Simulated multi-site Grid topology

We have assigned 3 computational resources to each Grid Site (for a total of 36 CRs). To reflect the use of different tiers in existing operational Grids, not all CRs are equivalent: the least powerful CR has two processors (which operate at the reference speed). A second class of CRs has four processors, and each processor operates at twice the reference speed. The third - and last - CR type contains 6 processors, each of which operates at three times the reference speed. Conversely, the least powerful type of CR is three times as common as the most powerful CR, and twice as common as the middle one (for a total of 18 reference CRs, 12 four-processor CRs and 6 of the most powerful CRs deployed in our simulated topology). It is assumed that all processors can be time-shared between different jobs.

We have assumed that storage resources offer "unlimited" disk space, but are limited in terms of access/write speed by the bandwidth of the link connecting the resource to the Grid Site. Each site has at its disposal exactly one such SR. Each site's data resource contains 6 out of 12 possible data sets. These data sets are distributed in such a way that 50% of the jobs submitted to a site can have local access to their needed data set.

C.6.2 Job parameters

We have used two different, equal-priority service classes (each accounting for half of the total job load) in our simulations; one is more data-intensive (i.e. higher data sizes involved), while the other is more CPU-intensive. At *each* Grid Site, two "clients" have been instantiated, one for each job type. Each client submits mutually independent jobs to its Grid Portal. All jobs need a single data resource

and a single storage resource. The ranges between which the relevant job parameters vary have been summarized in table C.1. In each simulation, the job load consisted of 2784 jobs. For each scheduling algorithm, we chose to use a fixed interval of $20s$ between consecutive scheduling rounds. From the arrival rates in

	CPU-Job	**Data-Job**
Input(GB)	0.01-0.02	1-2
Output(GB)	0.01-0.02	1-2
IAT(s)	30-40	30-40
Ref. run time(s)	100-200	40-60

Table C.1: Relevant service class properties

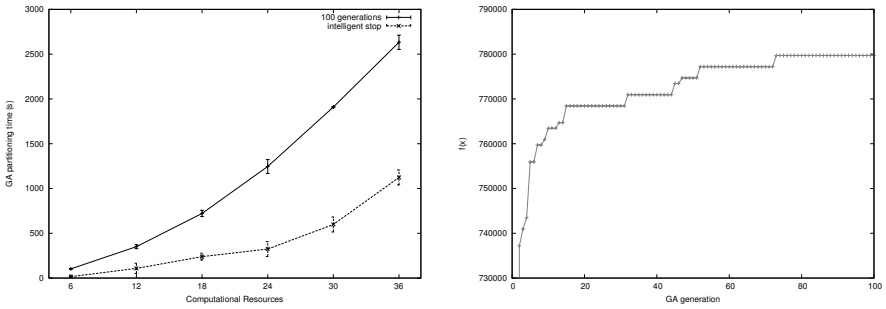
table C.1 and the fact that multiple sites submit job simultaneously, we are likely to find multiple jobs in the queue at the start of each scheduling round.

C.6.3 Comparison of DLT and GA based Partitioning

In general, our GA based partitioning strategy provides more functionality, as it is able to support different priority schemes, shared resources (service class 0 assignations) and local vs. foreign service differentiation. Its main drawback is the time needed to complete a GA run (with reasonable results); on our sample scenario, a naive stop condition of 100 generations takes on average $2632s$ ($26.32s$ per generation but it should be noted that this time is not exclusive for GA solution calculation, but is also spent on all other simulation tasks during partitioning) as can be seen in Figure C.10(a). More reasonable GA calculation times (with an average of $1123.4s$ can however be obtained when using a more intelligent stop condition (i.e. stop when over a period of 15 generations the cost function optimum changes by less than 0.5%). The DLT based approach on the other hand needs on average only $10s$. For the GA approach, we used Grefenstette's settings [23], with a population of 30 per generation, $\rho_C = 0.9$ and $\rho_M = 0.01$. In case faster partitioning times need to be attained, one can either tune GA parameters (smaller population sizes, faster stopping condition, etc.) or deploy a Service Monitor/Service Manager at every Grid Site, who are then responsible for communicating with the foreign site's Service Monitor components and partitioning the resources at their assigned site (as described in section C.3.3).

Figure C.10(b) shows the trend of the cost function optimum for different GA generations (partitioning occurred on the topology discussed in section C.6.1). The cost function used is the one discussed in section C.5.2.1 (Local Service CR partitioning) with Input Data Locality penalization. It is important to note that during the calculation of a resource-to-service partitioning, Grid operation does not stall

but continues as normal, as the Service Management components do not block any other management components.



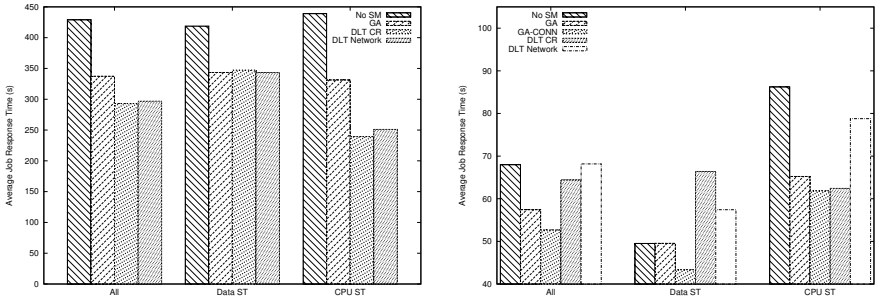
(a) solution calculation times (b) optimal fitness trend

Figure C.10: Genetic Algorithm measurements

C.6.4 Job response time

We define the *response time* of a job as the difference between its end time (time at which the job's final output block has been sent to the scheduler-assigned Storage Resource) and the time it is submitted to the scheduler. In figure C.11 we present this average job response time for different scenarios, comparing both the (DLT & GA based) Service Managed versus the non-Service Managed case (DLT CR attempts to minimize equation C.13 while DLT Network minimizes equation C.12 as explained in section C.5.1) while at the same time evaluating the different partitioning strategies discussed in previous sections for both network aware (see figure C.11(b)) and non network aware (see figure C.11(a)) scheduling algorithms. The results show that average job response times can be improved significantly (by 40.44% when non network aware scheduling is used and by 22.6% when network aware scheduling is employed) by employing a resource partitioning algorithm prior to scheduling. This behavior can be explained because resources are reserved for exclusive use by a service class. It is this service-exclusivity that forces the scheduler to not assign jobs to less-optimal resources (e.g. non-local access to needed input data, low processing power available,...), but to keep the job in the scheduling queue until a service-assigned resource becomes available.

It is noteworthy that the DLT based partitioning works best when network unaware scheduling algorithms are used (especially for the computationally intensive service class), as it outperforms the slower GA based partitioning strategies. However, when network aware scheduling strategies are employed (leading to much lower overall job response times as the scheduler takes into account the state of the network links interconnecting the various resources at the moment of scheduling),



(a) Non Network Aware scheduling (b) Network Aware scheduling

Figure C.11: Job response times

the GA based methods (particularly the GA based computational and GA-CONN computational/network resource partitioning algorithms) provide the best results.

If we compare the performance of the different GA based partitioning heuristics (see figure C.12) (note that when non network aware scheduling is employed, no connection partitioning results are shown, due to the fact that the non network aware scheduling algorithm does not take into account the connection reservation system) we notice that average job response times always improve when resources are partitioned amongst service classes. When scheduling non network aware, the best results are attained when using computational partitioning taking into account input data locality, as data intensive jobs can be run on computational resources reserved physically near resources that store much needed I/O data, leading in turn to less computational stalling, as I/O data suffers from less network bottlenecking. When network aware scheduling is employed, one is best of using a heuristic that partitions both computational and network resources. Network partitioning assures that service classes with high I/O requirements do not consume all bandwidth (thereby preventing computationally intensive service classes from retrieving their I/O), but instead force them to only use a predefined percentage of bandwidth.

C.6.5 Resource Efficiency

Using the same job load, the average hopcount over which data was transferred by data-intensive jobs (with hopcount equaling the amount of hops between data resource and computational resource added to the amount of hops between computational resource and storage resource) is shown in Figure C.13. We notice that average hopcount dropped by 4.8% when network unaware scheduling was employed (computational resource partitioning with data locality versus non-service partitioned resources), and by 5.5% when a network aware scheduling heuristic was used (network partitioning with data locality compared to the non-service man-

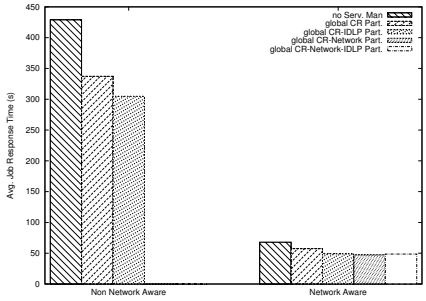


Figure C.12: Job response times for GA based partitioning heuristics

aged case), due to the fact that input/output data was located at resources closer to the job's service class' assigned CRs. Network resources are thus used most sparingly when computational and network resource partitioning with input data locality is employed together with a scheduling algorithm that takes into account the state of the network links interconnecting the job's resources.

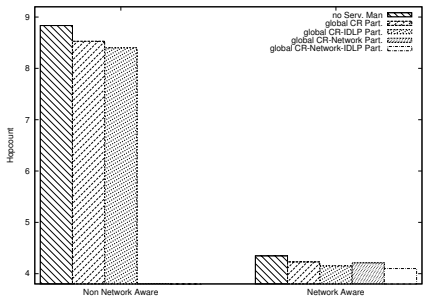


Figure C.13: Network resource efficiency

Furthermore, we calculated the average computational resource utilization:

$$\frac{\sum_{j \in Jobs_{CR}} Load_j}{Makespan \times speed_{CR}}$$

The improvement obtained by employing resource-to-service partitioning when using network unaware scheduling equals 17%, whereas in the case where network aware scheduling is used, it is 14.6%. Indeed, the fastest (and rarest in our topology) computational resources were automatically reserved for processing computationally complex jobs, disallowing data intensive jobs from cluttering these resources and using their full processing potential for those computationally intense jobs. The slower computational resources were then assigned to the data intensive service classes, who, because of their large I/O needs benefit more from having fast (i.e. LAN) access to much needed data.

C.6.6 Scheduling

We measured the time it takes to calculate a scheduling decision and noticed a decrease in scheduling time of 28.17% when comparing the service managed Grid to the non-service managed Grid in case network aware scheduling is used (i.e. from an average $7.88s$ in the non service managed case to 5.66s in the service managed Grid). This behaviour can be explained by the fact that a scheduler queries the Information Services for resources adhering to a job's requirements *and* assigned to either the job's service class or service class 0. When resources are partitioned amongst services, less results will be returned to the scheduler, allowing for faster schedule making decisions.

C.6.7 Priority - Service Class QoS support

In another experiment, we gave the CPU-intensive jobs higher priority than the data-intensive jobs and let the Service Manager construct a Virtual Private Grid (dedicated resource pool, scheduler and information service) for each service class. Due to the high priority of the CPU-intensive class, its cost function impact factor becomes higher which leads to more (and/or better) resources being assigned to the prioritized class. Also, during deployment of the VPG schedulers, the Service Manager configures the dedicated CPU-intensive scheduler to schedule those prioritized jobs as soon as possible, using a network aware scheduling algorithm (the data intensive jobs were also scheduled using a network aware scheduling algorithm, but were by default queued until the next scheduling round). The results are shown in figure C.14: the average job response time of the computationally intensive service class is substantially improved (due to more/better resources assigned to this service class and the ASAP scheduling policy enforced by the VPG scheduler), while the data intensive service class's average response time gets worse (prioritizing service classes over other service classes can not lead to win-win situations: the non-prioritized service classes' performance will deteriorate).

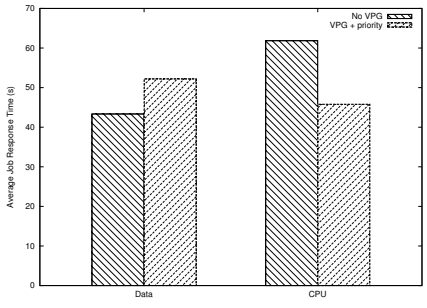


Figure C.14: VPG Service Class priority support

C.7 Conclusions

We proposed the use of a distributed service management architecture, following the OGSA 'service level manager' concept, capable of monitoring service characteristics at run-time and partitioning Grid resources amongst different priority service classes. This partitioning, together with the dynamic creation of per-service management components, lead to the introduction of the Virtual Private Grid concept. A variety of resource-to-service partitioning algorithms (some based on Divisible Load Theory and others employing Genetic Algorithm heuristics) were discussed and we evaluated their performance on a sample topology using NSGrid. Our results show that the proposed service management architecture improves both network and computational resource efficiency and job turnaround times, eases the process of making scheduling decisions, and at the same time offers service class QoS support. Management complexity and scheduling / information service scalability is improved due to the automated deployment of service class dedicated management components.

C.8 Acknowledgment

Bruno Volckaert and Marc De Leenheer would like to thank the Institute for the Promotion and Innovation through Science and Technology in Flanders (IWT-Vlaanderen). Pieter Thysebaert and Filip De Turck are research assistant and postdoctoral fellow, respectively, funded by the Research Foundation - Flanders (FWO-Vlaanderen).

References

[1] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. *Grid services for distributed system integration*. IEEE Computer, 35(6):37–46, 2002.

[2] *Enabling Grids for E-Science in Europe*. `http://egee-intranet.web.cern.ch`.

[3] I. Foster and al. *The Open Grid Services Architecture, Version 1.0*. draft-ggf-OGSA-spec-019 `http://forge.gridforum.org/projects/ogsa-wg`.

[4] K. Czajkowski and al. *The WS-Resource Framework Version 1.0*. `http://www.globus.org/wsrf/specs/ws-wsrf.pdf`.

[5] J.O. Kephart and D.M. Chess. *The Vision of Autonomic Computing*. IEEE Computer, 36(1):41–50, 2003.

[6] A.G. Ganek and T.A. Corbi. *The dawning of the autonomic computing era*. IBM Systems Journal, 42:5–18, 2003.

[7] B. Volckaert, P. Thysebaert, F. De Turck, P. Demeester, and B. Dhoedt. *Evaluation of Grid Scheduling Strategies through a Network-aware Grid Simulator*. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pages 31–35, 2003.

[8] K. Ranganathan and I. Foster. *Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids*. Journal of Grid Computing, 1:53–62, 2003.

[9] F. Berman et al. *Adaptive Computing on the Grid Using AppLeS*. IEEE Transactions on Parallel and Distributed Systems, 14:369–382, 2003.

[10] H. Casanova H. Dail, F. Berman. *A Decoupled Scheduling Aproach for Grid Application Development Environments*. Journal of Parallel and Distributed Computing, 63-5:505–524, 2003.

[11] R. Wolski, N. Spring, and Jim Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Future Generation Computer Systems, 15(5-6):757–768, 1999.

[12] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. *Grid Information Services for Distributed Resource Sharing*. In Proc. of the 10th IEEE International Symposium on High-Performance Distributed Computing, pages 181–194, 2001.

[13] I. Foster, A. Roy, and V. Sander. *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*. In Proceedings of the Eighth International Workshop on Quality of Service (IWQoS 2000), pages 181–188, 2000.

[14] A. Rodger. *Analyst report: Butler Group Subscription Services: Technology Infrastructure - IBM Tivoli Intelligent Orchestrator and IBM Tivoli Provisioning Manager*. `ftp://ftp.software.ibm.com/software/tivoli/analystreports/ar-orch-prov-butler.pdf`, 2004.

[15] H.L. Lee and al. *A Resource Manager for Optimal Resource Selection and Fault Tolerance Service In Grids*. In Proceedings of Cluster Computing and the Grid (CCGrid 2004), pages 572–579, 2004.

[16] M. Hovestadt, O. Kao, A. Keller, and A. Streit. *Scheduling in HPC Resource Management Systems: Queueing vs. Planning*. In Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, LNCS 2862/2003, pages 1–20, 2003.

[17] P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, and P. Demeester. *Network Aspects of Grid Scheduling Algorithms*. In 17th International Conference on Parallel and Distributed Computing Systems (PDCS'04), pages 91–97, 2004.

[18] D. Yu and T.G. Robertazzi. *Divisible Load Scheduling for Grid Computing*. In Proceedings of the IASTED 2003 International Conference on Parallel and Distributed Computing and Systems (PDCS), 2003.

[19] P. Thysebaert, F. De Turck, B. Dhoedt, and P. Demeester. *Using Divisible Load Theory to Dimension Optical Transport Networks for Computational Grids*. In Proceedings of OFC/NFOEC - on CD-ROM, 2005.

[20] Y. Kitatsuji, K. Kobayashi, Y. Kitamura, and al. *Deployment of APAN Tokyo XP and evaluation of source based routing*. Transactions of the Institute of Electronics, Information and Communication Engineers, J85-B:1164–1171, 2002.

[21] *LHC Computing Grid project*. `http://lcg.web.cern.ch/LCG`.

[22] D. Lu and P. Dinda. *GridG: Generating Realistic Computational Grids*. ACM SIGMETRICS Performance Evaluation Review, 40(4), 2003.

[23] J.J. Grefenstette. *Optimization of control parameters for genetic algorithms*. IEEE Trans. Systems, Man, and Cybernetics, 16-1:122–128, 1986.